UNIVERSIDADE FEDERAL DO PARANÁ

ALYSSON DE AVILA COSTA
FERNANDO ECKHARDT VALLE

# ABNTEXTOR: EDITOR DE DOCUMENTOS ACADÊMICOS

CURITIBA

2017

ALYSSON DE AVILA COSTA
FERNANDO ECKHARDT VALLE

# ABNTEXTOR: EDITOR DE DOCUMENTOS ACADÊMICOS

Trabalho de conclusão de curso apresentado ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Setor de Educação Profissional e Tecnológica da Universidade Federal do Paraná.
Orientador: Prof. Dr. Alexander Kutzke

CURITIBA

2017

## TERMO DE APROVAÇÃO

Alysson de Avila Costa
Fernando Eckhartdt Valle

**ABNTEXTOR: Editor de documentos acadêmicos**

Trabalho apresentado como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Federal do Paraná.

Curitiba, 07 de Dezembro de 2017.

BANCA EXAMINADORA

Orientador: Professor Dr. Alexander Robert Kutzke
SEPT/UFPR

Examinador: Professor Dr. Mauro Antônio Alves de Castro
SEPT/UFPR

Examinador: Professor Dr. Dieval Guizelini
SEPT/UFPR

# AGRADECIMENTOS

Agradecemos aos nossos familiares e amigos, por nos acompanharem e motivarem na trajetória acadêmica. Aos nossos professores e mestres por dividirem seus conhecimentos e experiência. Agradecemos especialmente ao nosso orientador Alexander Kutzke, por dividir conosco a alegria e satisfação de realizar esse trabalho.

Às nossas esposas, pela compreensão aos sacrifícios e dificuldades que enfrentamos, pelo carinho e apoio.

*“It’s a long way to the top*
*If you wanna rock ’n’ roll“*
*(Angus Young, Malcolm Young, Bon Scott; 1975)*

RESUMO

O AbnTeXtor é um editor de texto para trabalhos acadêmicos, levando em consideração primeiramente às exigências de forma da Associação Brasileira de Normas Técnicas (ABNT). O objetivo do trabalho é fornecer à comunidade acadêmica uma opção para escritas de trabalhos acadêmicos dentro das normas comumente exigidas, sem que o próprio usuário tenha que dominar todas elas. Para isso foi criado um editor de texto baseado em modelos de trabalhos previamente cadastrados, o qual gera marcação de texto em linguagem HTML de forma transparente para o usuário. Essa marcação é então convertida para linguagem LaTeX e compilada gerando o arquivo PDF que é entregue ao usuário final. O algoritmo que converte a marcação HTML para LaTeX também é um dos produtos do presente trabalho.

**Palavras-chaves**: LaTeX. AbnTeX. Edição de texto.

ABSTRACT

AbnTeXtor is a text editor to academic papers, considering the demands of the Brasilian Association of Technical Standards (Associação Brasileira de Normas Técnicas - ABNT). The objective of this work is to give to the academic community an option to academical paper writing within the most commom rules, despite user's knowledge about all of them. To accomplish this objective it was created a text editor based on templates previously registered, which generates HTML notation. This notation is converted to LaTeX and compiled, giving the final user the PDF file. The convertion algorithm is also one of the products of this work.

**Key-words**: LaTeX. AbnTeX. Text editor.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ABNT | Associação Brasileira de Normas Técnicas |
| abnTeX | ABsurdas Normas para TeX |
| DER | Diagrama Entidade Relacionamento |
| ERB | *Embedded RuBy* |
| IDE | *Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado |
| MVC | *Model-View-Controller* |
| SaaS | *Software as a Service*, ou Software como serviço |
| SQL | *Structured Query Language* |
| WYSIWYG | *What you see is what you get* |

SUMÁRIO

# 1 INTRODUÇÃO

A formatação de trabalhos de conclusão de curso, monografias, teses, dissertações e todo tipo de trabalho acadêmico tem normas extremamente rígidas, quase sempre baseadas nas regras da ABNT[1]. O estudante muitas vezes além de ter que fazer todas as suas pesquisas, análises e conclusões, deve em algum momento estar com as regras da ABNT em mãos quando chegar o momento de formatar seu trabalho.

Geralmente a formatação de trabalho acadêmico se faz usando as ferramentas de formatação de texto mais em voga no mercado, como o Microsoft Word ou o Libre Office (Open Office). Estes programas muitas vezes deixam a desejar no quesito confiabilidade - ou seja, não existe a confiança de que a formatação feita em determinado momento irá permanecer quando aberta em outro momento ou outra máquina, muitas vezes a simples inserção de uma imagem ou o aumento de um coluna em alguma tabela quebra a formatação, exigindo que muitos passos da formatação sejam revistos para fazer uma correção - e no quesito compatibilidade, ou seja, principalmente com o produto da Microsoft, a compatibilidade de um arquivo feito em determinada versão do software não irá abrir corretamente em outra versão[2].

Quem precisa fazer uma formatação mais avançada, como edição de fórmulas complexas de matemática, química ou física e de gráficos elaborados não pode recorrer aos editores citados acima devido a dificuldade de se fazer esse tipo de edição. Explica-se: geralmente esses editores, principalmente para fórmulas matemáticas, precisam de plugins ou módulos adicionos para ativar essas funcionalidades.

Para uma formatação mais precisa e muito mais controlada, os estudantes acabam recorrendo ao Latex, que é uma espécie de editor de texto. Porém este funciona na forma de linguagem de marcação (tal qual o HTML). O Latex, diferente dos editores como o Word, cheios de botões e menus intuitivos, e de fácil utilização, tem uma curva de aprendizagem muito elevada, muitas vezes criando, com isso, uma barreira para sua utilização.

Dessa forma, tem-se de um lado formatadores de texto visuais de fácil aprendizagem, porém que não conseguem lidar facilmente com formatações complexas, e de outro o Latex, um formatador extremamente poderoso, mas de difícil utilização.

A proposta desse projeto é justamente fazer a unificação desses dois mundos, uma interface de fácil utilização com todas as possibilidades e controle que um documento Latex oferece.

---

1 http://www.abnt.org.br/

2 https://superuser.com/questions/484524/microsoft-word-suddenly-lose-the-whole-formatting

O objetivo é a construção de um software, batizado de AbnTexTor, que será um editor e formatador de textos acadêmicos, com o funcionamento online, de fácil utilização e com capacidade para realizar formatações complexas sem que o usuário precise defini-las manualmente.

## 1.1 OBJETIVO GERAL

O objetivo desse trabalho é desenvolver o software AbnTexTor, que será um editor de texto online para trabalhos acadêmicos, capaz de abranger uma grande gama de formatações específicas definidas pelas normas da Associação Brasileira de Normas Técnicas (ABNT), sob a forma de *templates*, que uma vez escolhido pelo usuário, o isenta de customizar a formatação do trabalho - tamanho de letras, espaçamentos, margens, posição de imagens, quadros tabelas, e quaisquer outros elementos textuais.

## 1.2 OBJETIVOS ESPECÍFICOS

- Analisar as opções disponíveis no mercado para a escrita e formatação de trabalhos acadêmicos;
- Construir um editor de texto para fácil manipulação de trabalhos acadêmicos e posterior conversão para linguagem Latex;
- Disponibilizar uma infraestrutura mínima para o editor de texto, com controle de acesso, cadastro de trabalhos, usuários e regras de formatação;
- Disponibilizar o algorítimo de conversão de tags HTML para Latex;
- Documentar o sistema.

## 1.3 PROBLEMA

Ao tratar de um editor de texto, inevitavelmente nos deparamos com o fato de que existe uma grande quantidade de opções disponíveis no mercado, tanto em lojas especializadas em informática quanto na internet. A escolha de uma ou outra opção pode ser influenciada por vários motivos, desde preço, funcionalidades, gosto ou filosofia pessoal. Trata-se no capítulo 2 algumas dessas opções, trazendo os pontos positivos e negativos de cada software analisado.

O ambiente acadêmico, por sua vez, caracteriza-se por uma intensa produção de materiais escritos - da graduação à pós-graduação. Quanto maior a titularidade postulada, mais atenção é dada aos quesitos formais desses materiais, sejam artigos, monografias, dissertações, teses entre outros. Os acadêmicos brasileiros precisam, portanto, de ferramentas para que possam dar vazão à produção técnica e científica.

Conciliar a produção acadêmica de textos com as ferramentas disponíveis, contudo, nem sempre é uma tarefa simples. Não se encontra uma única opção que isentasse o usuário de normas técnicas, como alinhamento, bordas, espaçamento, fontes, posição de tabelas, quadros, figuras, etc. Todas as opções fazem com que o usuário tenha que dar conta não só do conteúdo, mas também da forma do material que está produzindo.

Quando a ferramenta é especialista em formatação, isso vem ao custo de uma enorme curva de aprendizagem de comandos e marcações (tags) e detalhes técnicos que muitas vezes afastam usuários iniciantes.

Por fim, quando uma opção é de utilização intuitiva, em nuvem e permite que o usuário produza seu conteúdo sem se preocupar com as regras técnicas e formais do trabalho, ela tem um custo elevado e muitas vezes com uma forma inadequada de cobrança ou período de acesso.

Muitos dos problemas citados acima são revisitados no capítulo 2, em uma análise minuciosa dos softwares de edição de texto.

## 1.4 JUSTIFICATIVA

O software produzido pelo presente trabalho busca dar conta de todos os elementos abordados anteriormente ao entregar um software para formatação de trabalhos acadêmicos nos padrões brasileiros mais comuns, isto é, os definidos pela Associação Brasileira de Normas Técnicas (ABNT).

Um dos pontos centrais do software é o de que a maior parte da formação estará implícita na escolha do *template*, feita pelo usuário ao iniciar um novo trabalho. Ao usuário será dada apenas as opções de estilo mais comuns, enquanto elementos estruturais e demais exigências formais serão trabalhadas pelo próprio software.

O software será disponibilizado de forma online, sendo possível acessá-lo de qualquer dispositivo com acesso à internet. E, por fim, para que seja efetivamente uma contribuição à comunidade acadêmica brasileira é importante que seu acesso seja no mínimo gratuito - ou melhor ainda, como nesse caso, livre.

## 1.5 ESTRUTURA DO DOCUMENTO

Além do presente capítulo, este documento apresenta os seguintes conteúdos: o Capítulo 2 embasa teoricamente o trabalho e algumas decisões tomadas no seu desenvolvimento, discutindo o Latex, o software livre e as principais opções de editores de texto encontradas atualmente.

O Capítulo 3 detalha a metodologia de desenvolvimento do projeto e os ma-

teriais (documentos, diagramas, ferramentas) produzidos, bem como apresenta o cronograma seguido pelo equipe, tecnologias e ferramentas utilizadas.

O Capítulo 4 apresenta o sistema desenvolvido em si, sua arquitetura e funcionamento, telas, funcionalidades e as responsabilidades de cada tipo de usuário.

O Capítulo 5 faz a conclusão do trabalho, abordando as principais falhas e dificuldades encontradas ao longo do seu desenvolvimento, as possíveis melhorias e caminhos que podem ser tomados para além da sua conclusão.

O apêndice contém os artefatos e materiais referidos nos capítulos anteriores, de protótipo de telas à diagramas e especificações UML.

# 2 FUNDAMENTAÇÃO TEÓRICA

O presente capítulo apresenta alguns pontos de discussão que tangenciam o desenvolvimento do software proposto e do presente trabalho. Inicia-se com a exposição do Tex e Latex - o primeiro, um sistema tipográfico sofisticado e o segundo, um conjunto de instruções de alto nível para o primeiro. Esses dois sistemas estão no motor do AbnTexTor. Neste capítulo também se discute brevemente alguns modelos de licenciamento de software. Por compreender a importância de gerar um produto que atenda à comunidade e que a ela pertença, é apresentado o tipo de licenciamento da aplicação. Algumas alternativas à criação e formatação de trabalhos acadêmicos (objetivo do software desenvolvido neste trabalho) são apresentados, com seus pontos positivos e negativos, dando a dimensão de que meio o AbnTexTor está inserido.

## 2.1 LATEX

O Latex é um conjunto de macros usados para a programação e diagramação de textos utilizando a linguagem Tex. O Latex foi desenvolvido na década de 80 pelo matemático Leslie Lamport. É utilizado amplamente no meio acadêmico e científico para a escrita de teses, monografias e artigos devido a alta qualidade tipográfica e fácil manipulação para escrever fórmulas, equações e todo tipo de conteúdo da principalmente na área de exatas.

A principal ideia por trás do Latex é manter o máximo possível uma distância entre o autor e a apresentação visual de seu documento, fazendo dessa forma que ele (o autor) foque o toda sua atenção apenas em seu pensamento e raciocínio para desenvolver o conteúdo.

Diferentemente dos editores do tipo WYSIWYG (What You See Is What You Get - O que você vê é o que você terá), no Latex o formato definitivo do texto só será conhecido após sua finalização, no caso, a compilação do documento.

Por trás do Latex tem-se o Tex, que é um sistema de tipografia criado por Donald Knuth em 1978 depois que ele achou terrível a tipografia da segunda edição de seu livro *The Art of Computer Programming*. A primeira edição do livro foi lançada em 1968 e foi impressa em tradicional método de prensas com tipos de metal do século XIX (*Hot metal typesetting*), enquanto o segundo volume, de 1976, foi produzido em um dos primeiros sistemas que os eliminaram (*Phototypesetting*). O desapontamento de Knuth com a nova tipografia e seu interesse pela tipografia digital formaram a base para a criação do Tex.

O nome Tex foi escolhido por Knuth por representar em sua raíz grega tanto

o termo "tecnologia"quanto "arte"(KNUTH, 1986, p. 1), o que pode nos dar a exata dimensão do seu propósito. O Tex está atualmente na versão 3.14159265 (*sic*), lançada em 2014. A versão 3 foi lançada em 1989 após uma reescrita total do código fonte.

As principais vantagens do Latex são:

- Criação de documentos com aparência profissional.
- O Latex encoraja as pessoas a pensarem no conteúdo e na distribuição lógica desse conteúdo e não na aparência, o resultado disso são textos extremamente bem estruturados.
- Estruturas mais complexas de formatação como bibliografias, índices, citações, notas de rodapé e tabelas podem ser gerados facilmente e de forma consistente no documento.
- A edição de fórmulas matemáticas é extremamente robusta e flexível.
- Tanto o Latex quanto o Tex são softwares livres.

Devido a essa alta capacidade de formatação gráfica proporcionada pelo Latex, é ele que está no núcleo de funcionamento da aplicação aqui proposta: a robustez, qualidade, tipo de licenciamento e facilidade de uso foram os fatores fundamentais para sua escolha.

## 2.2 SOFTWARE LIVRE

A aplicação desenvolvida neste trabalho é um software livre. Um software para ser considerado livre, de acordo com as indicações que a *Free Software Foundation*, deve seguir os preceitos das 4 liberdades fundamentais[1]:

- A liberdade de executar o programa como você desejar, para qualquer propósito (liberdade 0).
- A liberdade de estudar como o programa funciona, e adaptá-lo às suas necessidades (liberdade 1). Para tanto, acesso ao código-fonte é um pré-requisito.
- A liberdade de redistribuir cópias de modo que você possa ajudar ao próximo (liberdade 2).
- A liberdade de distribuir cópias de suas versões modificadas a outros (liberdade 3). Desta forma, você pode dar a toda comunidade a chance de beneficiar de suas mudanças.

[1] 4 liberdades retiradas do site do projeto GNU: https://www.gnu.org/philosophy/free-sw.pt-br.html

A essência do software livre reside nos conceitos de liberdade e não de preço grátis. Richard Stellman no seu famoso manifesto *Why Open Source misses the point of Free Software* já disse que a ideia de software livre “é uma questão de liberdade, não de preço - pense em “liberdade de expressão”, não em “cerveja grátis”. (STALLMAN, sem data)

O ponto central da questão de liberdade de software diz respeito a um estado muito mais amplo do que simplesmente liberar um produto de software. Ainda de acordo com Stellman as liberdades do software são importante pois

> [...] são essenciais não apenas para os propósitos individuais dos usuários, mas para a sociedade como um todo, pois elas promovem solidariedade social — isto é, compartilhamento e cooperação. Elas se tornam ainda mais importantes à medida que nossa cultura e atividades cotidianas se tornam mais digitalizadas. Num mundo de sons, imagens e palavras digitais, o software livre se torna essencial para a liberdade em geral. (STALLMAN, sem data)

Existem várias licenças de software livre, as 3 habitualmente mais usadas são a GPL, a MIT e a APACHE.

A licença GPL, tanto a versão 2 quanto a 3, é uma licença do tipo "copyleft", isso faz com que quem distribui o seu código ou uma obra derivada dele, deve disponibilizar a fonte com o mesmo tipo de licença. Ou seja, um software derivado de alguma aplicação com licença GPL, também será GPL. Alguns exemplo de softwares que usam a GPL: Linux, Git e Wordpress.

Já a licença MIT é do tipo permissiva e concisa. Ela permite que quem tenha o código com essa licença faça o que quiser dele, inclusive fechá-lo, desde que forneça uma atribuição de volta ao dono original e não lhe responsabilize. Exemplos de softwares com essa licença: jQuery e Rails.

A licença Apache, tal qual a MIT, também é permissiva, mas tem uma concessão expressa de direito de patentes dos contribuintes para os usuários. Exemplos: Apache, SVN e NuGet.

A licença escolhida para o projeto foi a GPL, essa licença incorpora totalmente as 4 liberdades. O grande diferencial dela é que com essa licença qualquer software gerado a partir de um outro software com a licença GPL também deverá ter licença GPL. As outras licenças são um pouco mais flexíveis nesse ponto, permitindo inclusive que o software tenha seu código fechado ou use o software livre para fazer parte de um software proprietário. Como nesse projeto existe uma preocupação social, então escolhe-se a licença que de total liberdade para qualquer pessoa usar e modificar

e fazer o uso que quiser da aplicação desde que não restrinja o acesso de suas modificações e melhorias, caso resolva disponibilizar ao mundo.

Existem vários pontos positivos em fazer um software livre. Um software que seja livre tem a possibilidade quase infinita de melhorias, pois vários usuários poderão mexer diretamente no código para fazer as alterações necessárias, essa alterações se forem realmente interessantes, poderão ser embutidas no código fonte original. Um software livre tem o mundo inteiro de usuários como colaboradores de melhoria da aplicação, podendo ser desde usuários do sistema que relataram um bug, programadores que fizeram alterações ou tradutores, que ajudaram no projeto traduzindo o programa para sua língua local, aumentando assim a possibilidade de mais usuários entenderem o programa.

Existem grandes aplicações que são software livre, grande parte dos servidores web que fazer a internet funcionar hoje em dia são softwares livres, o sistema que faz os smartphones Android rodar é software livre e as melhores linguagens de programação que temos são software livre.

A escolha por deixar a aplicação livre foi feita justamente pela ideologia do software livre, compartilhar um software para que mais pessoas se beneficiem, de alguma maneira, fazendo uso dele, essa é a questão principal dessa escolha nesse projeto.

## 2.3 O MERCADO DE EDITORES DE TEXTO

Existe hoje no mercado e na internet uma grande variedades de programas para edição de texto. Eles variam em grande medida nas interfaces e funcionalidades apresentadas. Alguns apresentam opções mais ricas de edição e editoração de textos, enquanto outros são focados em marcação semântica, desenvolvimento de código e linguagens de programação. Apresenta-se alguns deles a seguir, dividindo-os entre softwares tradicionais, softwares online e editores de documentos Latex/Tex.

### 2.3.1 SOFTWARES TRADICIONAIS

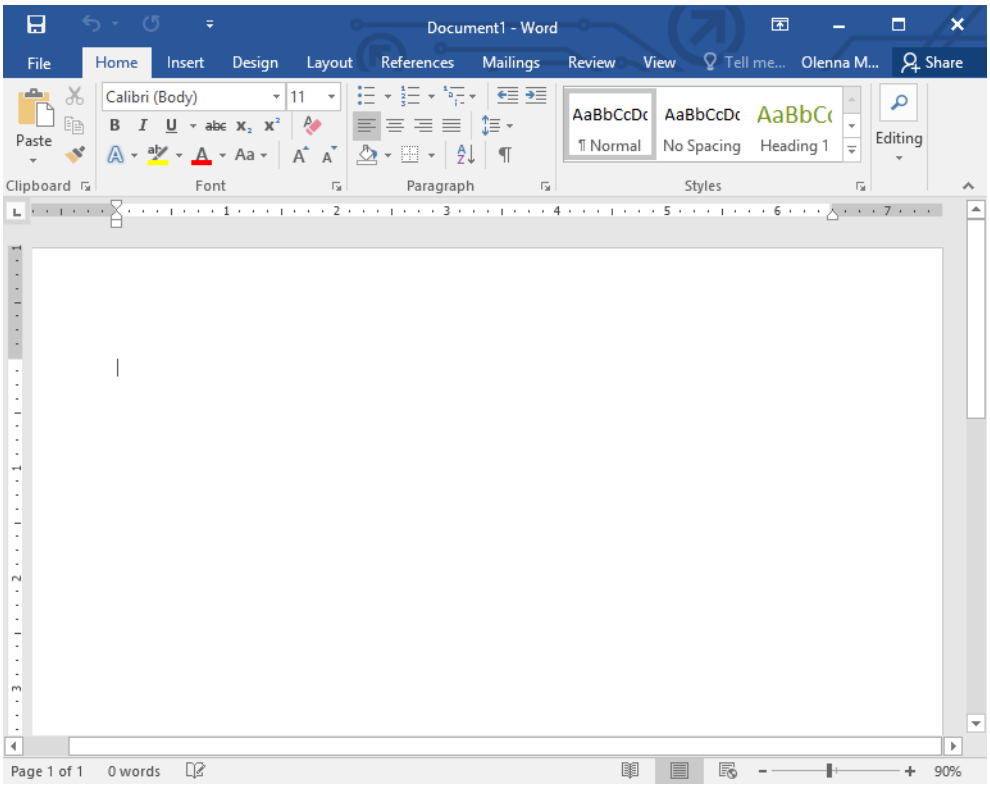
Usualmente a formatação de trabalhos acadêmicos é feita usando softwares gráficos conhecidos no mercado. Os dois programas de desktop mais conhecidos que existem são o Microsoft Word e o LibreOffice Writer.

#### 2.3.1.1 MICROSOFT WORD

O Microsoft Word foi criado em 1983 para rodar no DOS. O preceito básico do programa é a facilidade de uso. Tendo menus, abas e acessos intuitivos a curva de aprendizagem acaba tornando-se muito baixa. A operação básica do programa é

aprendida rapidamente. Trata-se do mais tradicional e conhecido software para edição de textos, dominante no setor privado e em boa parte dos computadores com o sistema operacional Windows, em que pese o preço em cifras elevadas para compra de licenças pessoais do mesmo. Esse é um dos principais empecilhos para o uso legal do software, ainda que a Microsoft forneça uma versão gratuita, temporária e limitada do pacote de aplicativos Office com computadores novos vendidos com seu sistema operacional.

Figura 1 – Editores de texto: Microsoft Word

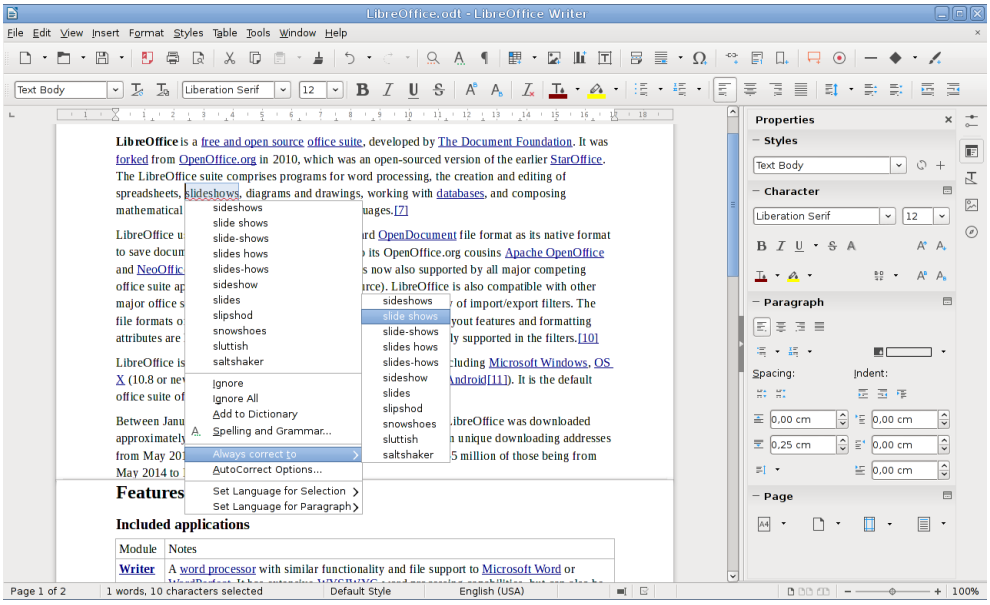



Fonte: https://goo.gl/NDmTKN

#### 2.3.1.2 LIBREOFFICE WRITER

Outro programa bastante difundido para formatação de textos é o LibreOffice Writer, programa similar ao Word, porém livre e gratuito. Também é um programa que preza pela facilidade de uso e assim como o Word tem uma curva de aprendizado bastante curta. Apresenta interface e funcionalidades muito semelhantes e é uma opção muito comum para usuários de softwares gratuitos.

Figura 2 – Editores de texto: Libreoffice Writer




Fonte: https://goo.gl/77Zkzw

Esses programas têm como grande atrativo, a facilidade de uso, porém um ponto que ainda deixa a desejar é justamente a formatação mais elaborada, mais rebuscada, onde regras devem ser seguidas rigorosamente. A probabilidade de quebra de formatação, seja por problemas de fonte, ou de versões do software, ou ainda problemas de natureza indefinida não são pequenas nesses programas.

A facilidade em mexer nessas aplicações é grande, porém a confiança na robustez que eles apresentam para garantir sua formatação não é tão grande assim, deixando para o usuário toda a responsabilidade e fazendo com que ele se preocupe com detalhes que não deveriam lhe preocupar.

### 2.3.2 SOFTWARES ONLINE

Além dos softwares para desktop, o que existe hoje também são os softwares de formatação e edição de texto na nuvem. Como principais expoentes desse nicho tem-se o Google Docs e o Microsoft Word 365. Ambos não requerem a instalação no computador e são acessados pela internet via navegador.

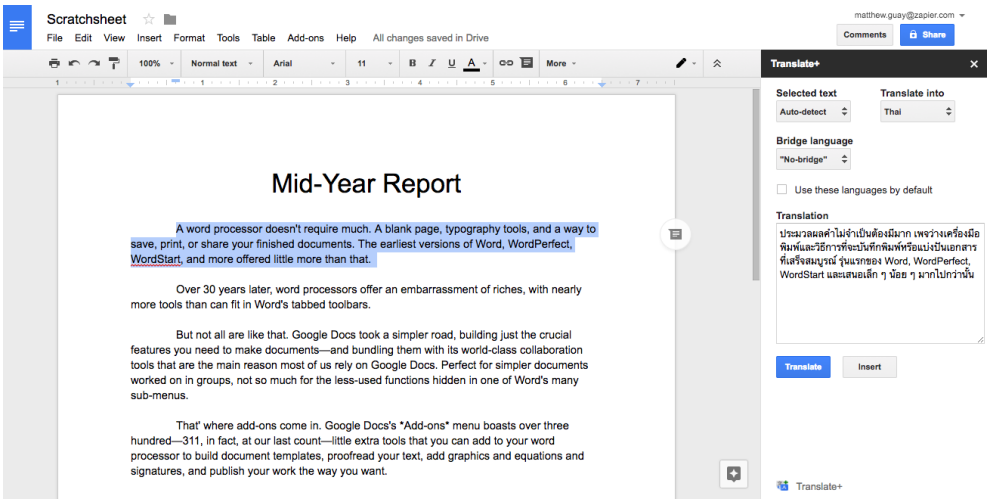
#### 2.3.2.1 GOOGLE DOCS

Desenvolvido e mantido pelo Google, foi um dos primeiros softwares desse gênero a se popularizar. Oferece criação e hospedagem de arquivos de textos editados diretamente do navegador. Apresenta uma boa gama de opções e funcionalidades para formatação, sendo uma alternativa viável para escrita, formatação e compartilhamento

de trabalhos acadêmicos. Permite a edição simultânea de documentos e a exportação para diferentes formatos.

Apesar da sua popularidade e versatilidade, o Google Docs não oferece nenhuma assistência para trabalhos acadêmicos. A formatação de um documento criado é sempre genérica e generalista, devendo o usuário fazer todas as alterações para que o documento se enquadre em normas técnicas.

Figura 3 – Editores de texto: Google Docs

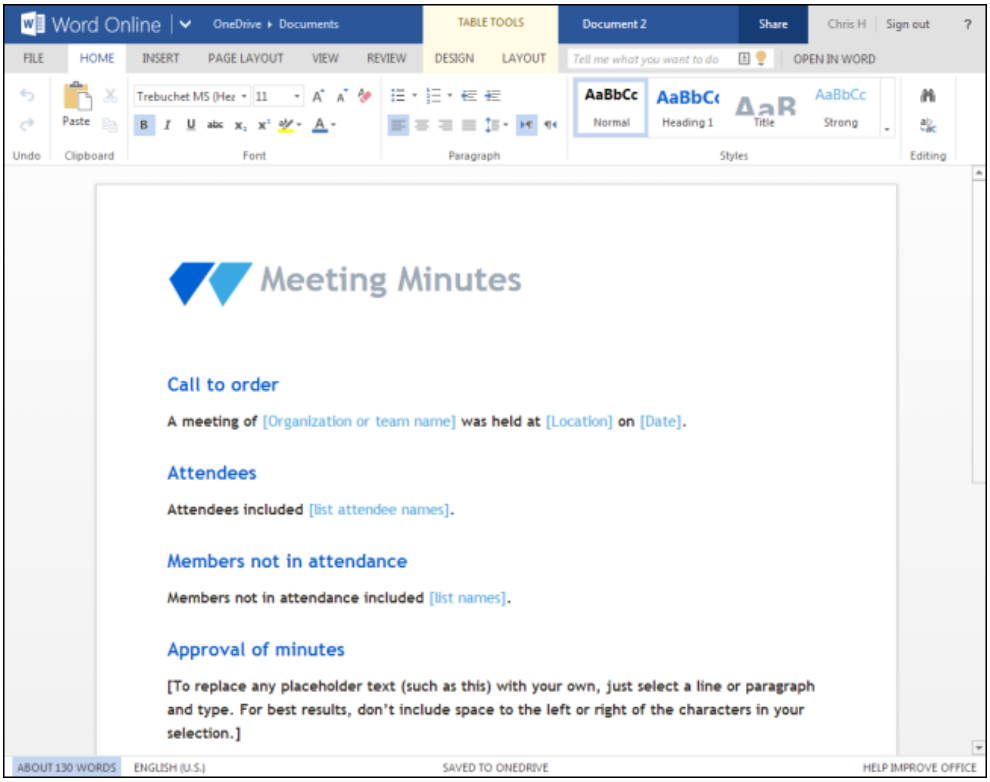



Fonte: https://goo.gl/QWftwa

#### 2.3.2.2 MICROSOFT OFFICE 365

Trata-se da opção online do já apresentado Microsoft Office. Conta com as vantagem dos editores de texto em nuvem, notadamente a edição simultânea de documentos, e é disponibilizado gratuitamente. Mas como em sua versão *desktop*, não tem recursos, funcionalidades ou assistência para formatação de trabalhos acadêmicos de forma mais específica.

Figura 4 – Editores de texto: Microsoft Word 365




Fonte: https://goo.gl/chDn4A

Mais uma vez a formatação mais elaborada de documentos é um ponto que deixa a desejar nessas opções online. Apresentam recursos de compartilhamento interessantes, mas possuem menos opções que suas contrapartes de desktop. A formatação de um documento onde regras rígidas devem ser seguidas, principalmente em relação à forma, não é apropriada em editores online de natureza generalista.

### 2.3.3 EDITORES LATEX

Um ponto negativo em comum dos editores apresentados anteriormente é a ausência de assistência para a formatação de trabalhos em normas técnicas, os itens a seguir tratam especificamente de editores Latex, que podem suprir essa necessidade.

#### 2.3.3.1 PLUGINS PARA EDITORES DE TEXTO

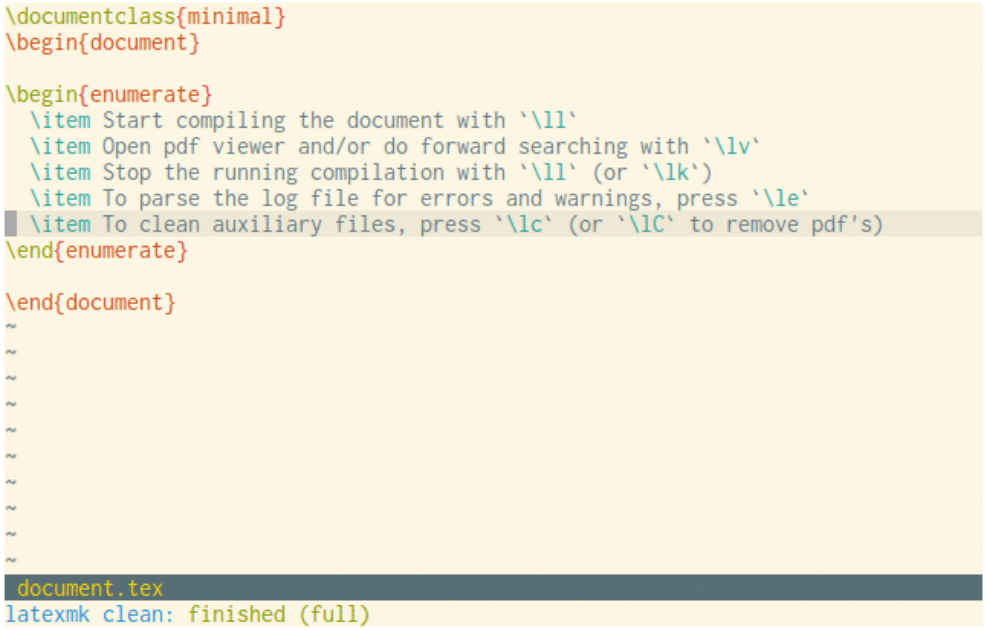
Por padrão, qualquer editor de texto é suficiente para editar um documento usando a formatação Latex, porém é necessário um ambiente de compilação e conhecimento apurado da linguagem Tex para se construir um documento dentro de regras definidas.

Os editores Latex que existem atualmente facilitam bastante a vida do usuário no quesito usabilidade, porém ainda não existe um editor com um ambiente gráfico

integrado que seja eficiente para utilizar todo o potencial que o Latex dispõe para a formatação de texto. O usuário ainda precisa dominar a sintaxe da linguagem Tex para poder usufruir plenamente de um desses editores.

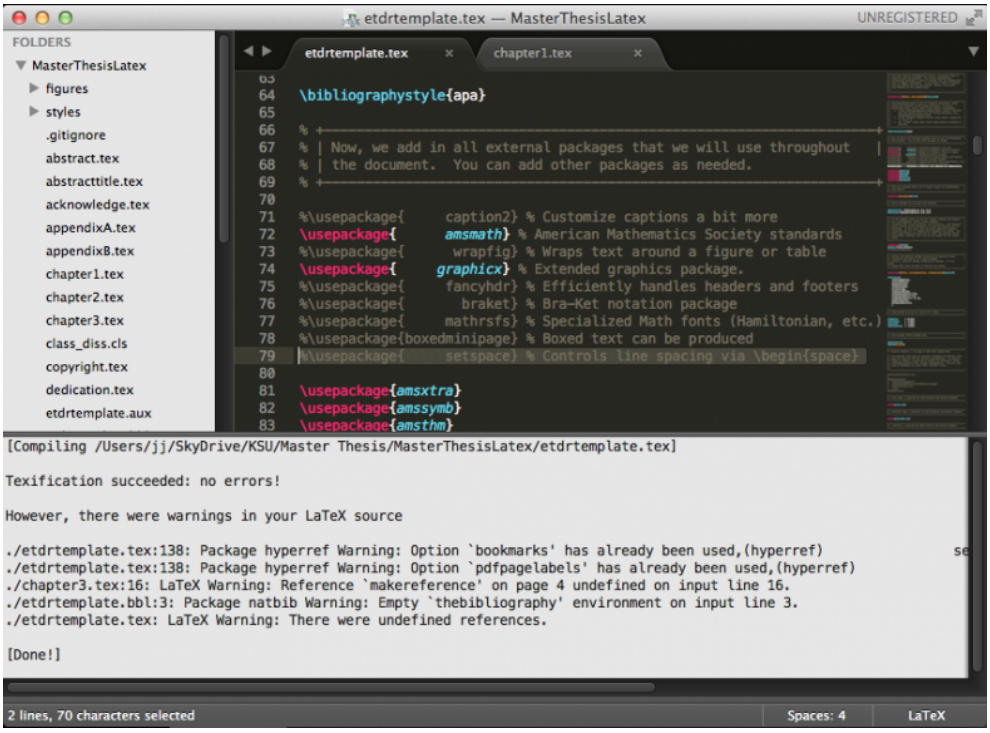
A maioria dos mais robustos editores de texto (para programadores) que existem no mercado tem plugins que auxiliam na manipulação de arquivos escritos em Latex. Para citar alguns: tem-se o *Sublime Text* com o plugin *LateXTools*, o *Vim* com o plugin *vimtex* e também existe ainda o *VSCode* com o plugin *LaTeX Workshop*.

Figura 5 – Editores de texto: Vim com o plugin vimtex



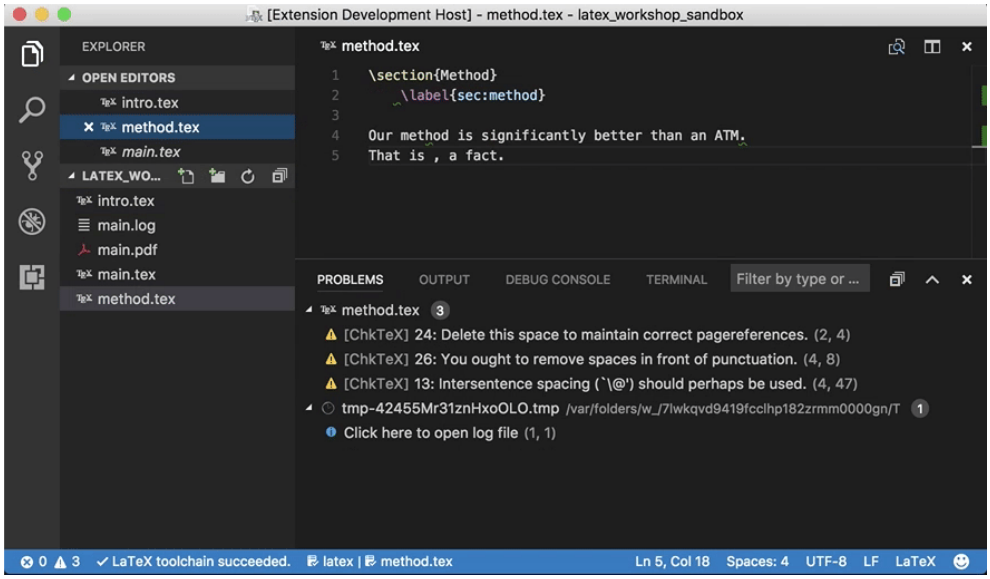

Fonte: https://github.com/lervag/vimtex

Figura 6 – Editores de texto: Sublime com plugin LatexTools




Fonte: https://goo.gl/oUfYPe

Figura 7 – Editores de texto: VS Code com o plugin Latex Workshop




Fonte: https://github.com/James-Yu/LaTeX-Workshop

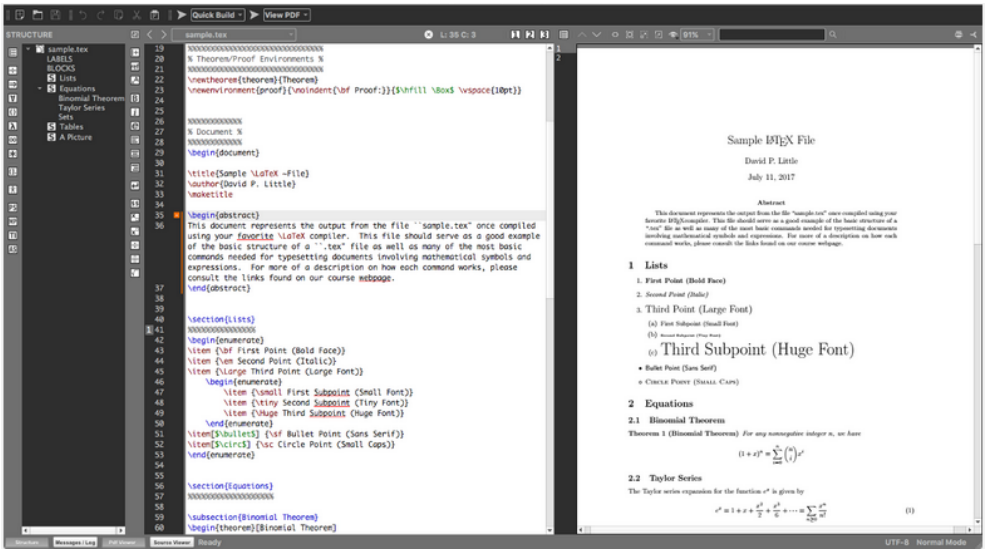
#### 2.3.3.2 TEXMAKER, TEXSTUDIO

Hoje ainda existem editores amigáveis de Latex, que são verdadeiras *IDE*s para manipular documentos com a linguagem. As duas ferramentas abaixo são editores

Tex multiplataforma.

O Texmaker é um dos destaque no quesito qualidade de seus componentes visuais. É o mais polido software de edição Latex dentre os apresentados até momento. Seu desenvolvimento teve início em 2003 e atualmente está na versão 5.0.2.

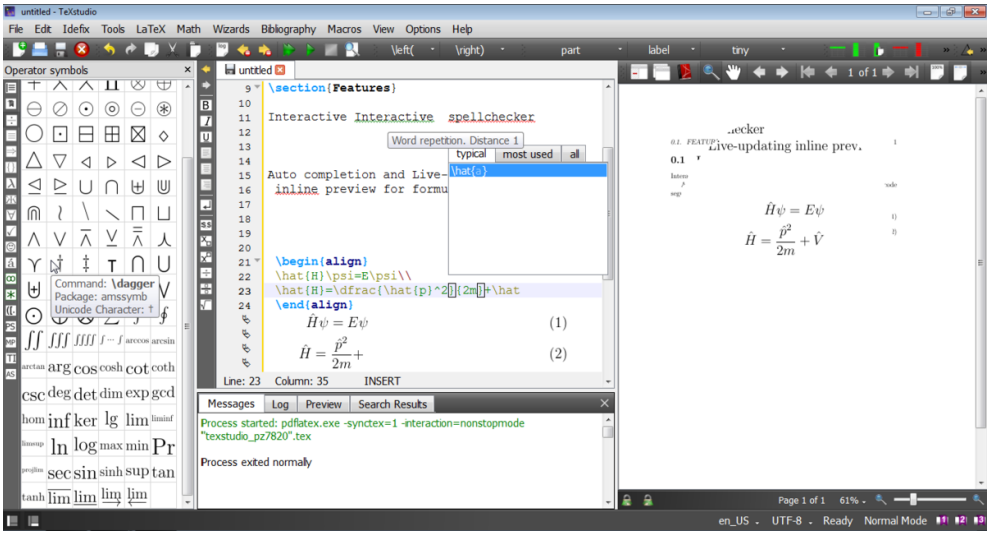
Figura 8 – Editores de texto: TeX Maker



Fonte: https://goo.gl/rP24Fv

O TexStudio é um projeto derivado do Texmaker iniciado em 2009 e é um dos editores mais amigáveis para a utilização da linguagem Latex, embora tenha muitos botões que façam as formatações e ajustes desejados do texto, ainda é um editor para escrever conteúdo e código, obrigando o usuário a dominar, pelo menos minimamente, a linguagem Tex.

Figura 9 – Editores de texto: TeX Studio




Fonte: https://goo.gl/wJqhEH

Os dois editores apresentados são muito parecidos, haja vista o segundo ser um *fork* do primeiro. Segundo o site do TexStudio essa divisão aconteceu por conta de diferenças de filosofia sobre o funcionamento e funcionalidades do software, além do "processo de desenvolvimento não aberto"do Texmaker (https://www.texstudio.org/).

#### 2.3.3.3 LATEX EDITOR, TEXNIC CENTER, WINEDT

Assim como os editores anteriores, existem ainda outras opções exclusivas para o sistema operacional Microsoft Windows disponíveis na internet - algumas mais antigas, outras mais recentes, mas em geral todas gratuitas.

O Latex Editor teve sua primeira versão beta lançada em janeiro de 2005. Sua versão mais recente continua em formato beta sob a versão 0.53 e data de outubro de 2009. Há quase 10 anos sem atualização, dificilmente ocorrerão evoluções desse software. Sua página, porém, continua funcional e é possível baixar o instalador bem como os pacotes de integram o programa.

Figura 10 – Editores de texto: Latex Editor

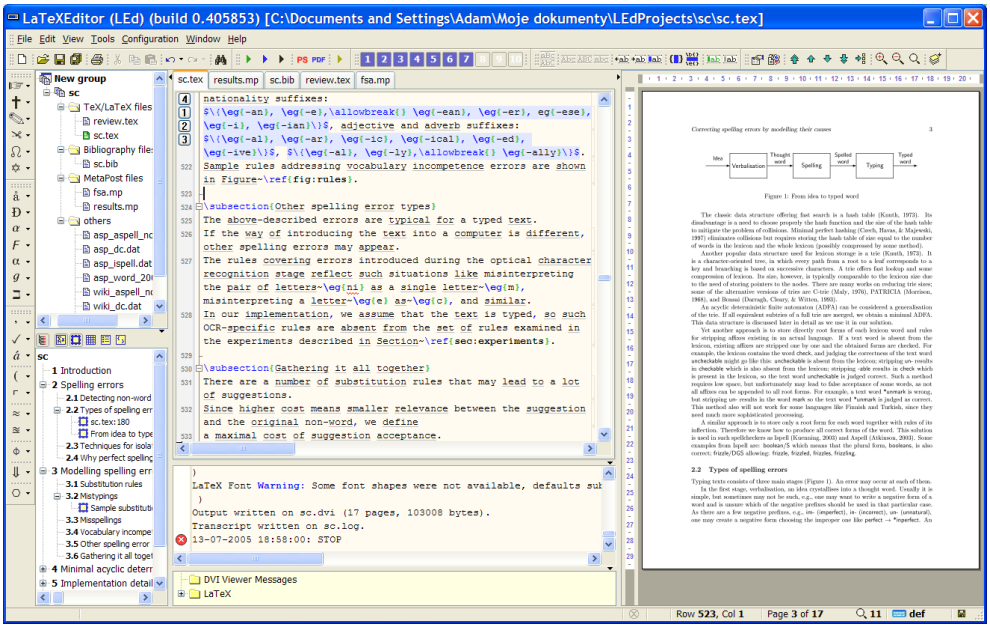
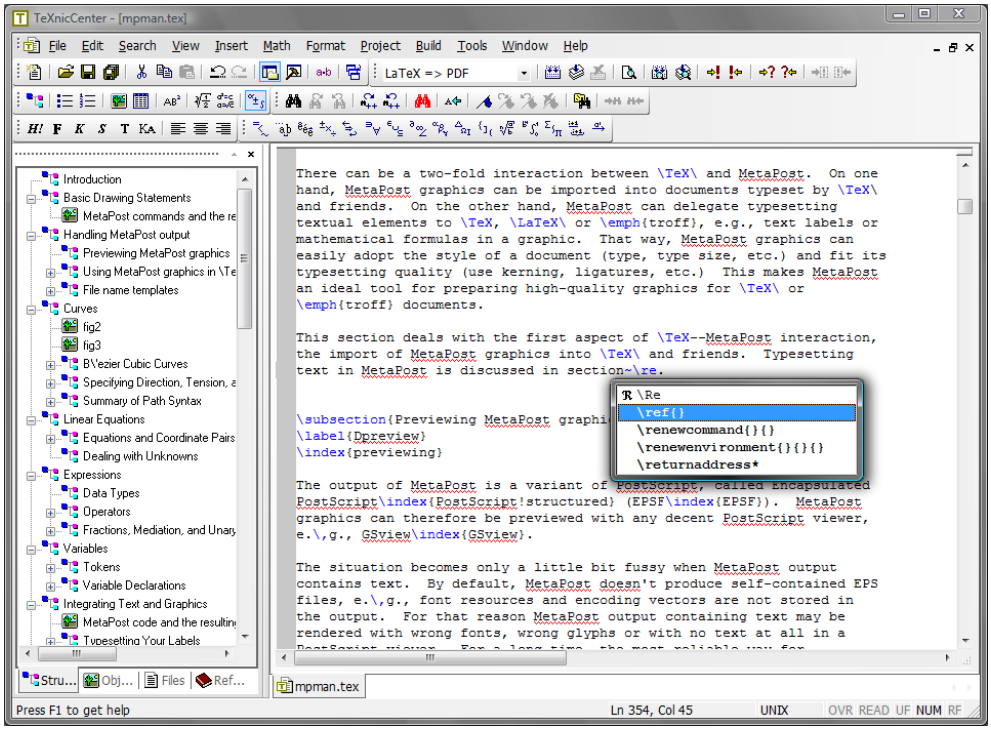



Figura 11 – Fonte: http://www.latexeditor.org/screens/main$_{l}arge.png$

O TeXnic Center se apresenta como um "ambiente de documentação integrado"aproveitando o acrônimo de IDE em inglês. Reúne todas as ferramentas necessárias para criar, escrever, compilar e visualizar documentos Latex. Possui versões em 32 e 64-bits disponíveis diretamente no site do programa, atualmente sob a versão 2.02. Apesar de possuir "todas"as ferramentas necessárias, porém, é necessário instalar uma distribuição Tex e um visualizador de PDF.

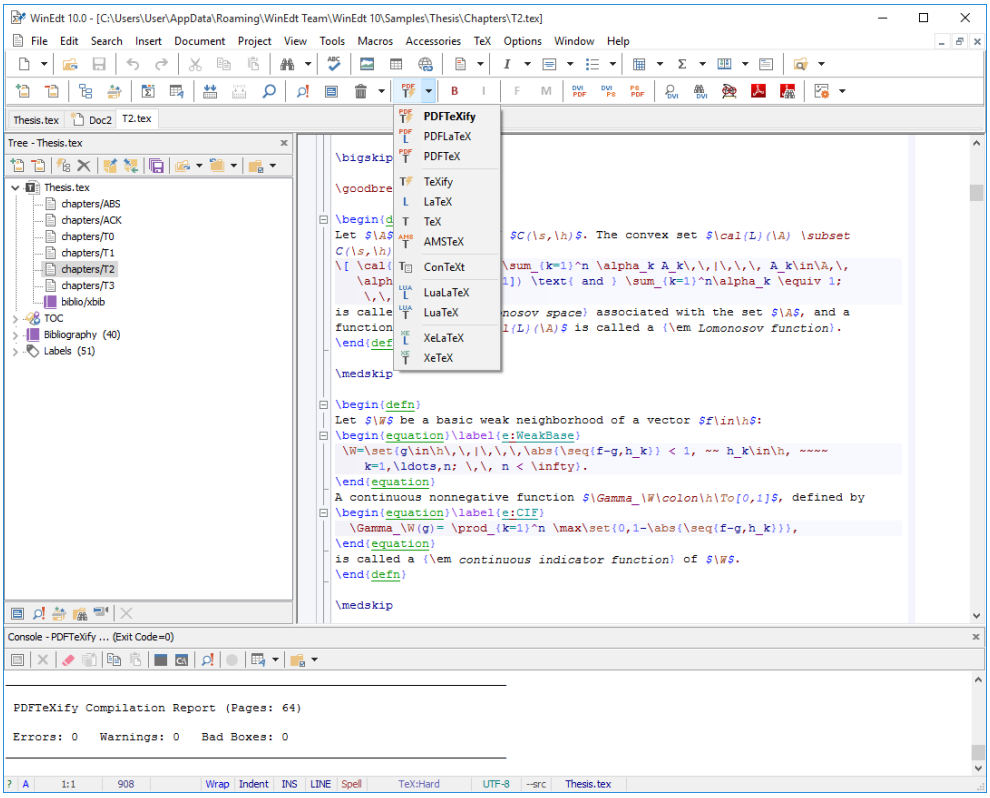
Figura 12 – Editores de texto: TeXnic Center




Fonte: https://goo.gl/4nC7tG

O WinEdt é um software que tem sido desenvolvido desde meados dos anos 1990. Está em sua versão 10, otimizada para o sistema operacional Windows 10. Experimentou um breve período como software pago (shareware) em 1995, mas, como relata o desenvolvedor no site do programa, apenas cinco cópias foram registradas naquele ano, incluindo a dele próprio (http://www.winedt.com/about.html).

Figura 13 – Editores de texto: WinEdt




Fonte: https://goo.gl/3j5LS5

Encerra-se a apresentação dessa seção com WinEdt pela notória consideração feita pelos seus autores no site da aplicação: o maior desafio encontrado pelos desenvolvedores atualmente é tornar o software mais acessível para usuários intermediários, que cogitam elaborar interfaces e tutoriais que ajudem novos usuários no futuro. (http://www.winedt.com/)

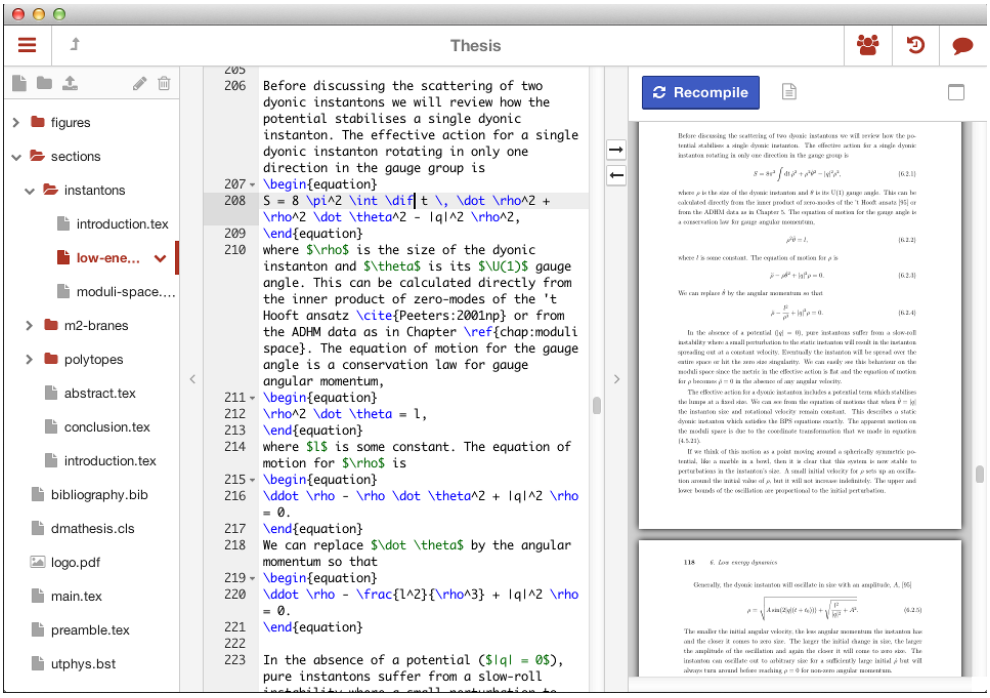
#### 2.3.3.4 SHARELATEX

A opção mais famosa para edição de arquivos Tex na nuvem é o Sharelatex (atualmente integrado com o Overleaf), que nada mais é que um ambiente de desenvolvimento e compilação do Latex na nuvem. Tal qual outros editores de texto que estão na nuvem, a grande vantagem do Sharelatex é a possibilidade de compartilhar e editar arquivo ao mesmo tempo com outro usuário. Ainda, com o Sharelatex é dispensável a configuração de um ambiente para fazer a compilação dos arquivos.

O Sharelatex é uma aplicação open source, ou seja, seu código é disponível para quem quiser utilizar e implementar sua própria versão da aplicação. Embora tenha um site onde o usuário possa logar e pagar por alguns *features* do produto, a aplicação é disponibilizada no github para qualquer pessoa gerar seu próprio serviço.

Apesar do poder para formatação de trabalho e das funcionalidades de compartilhamento e edição online de trabalhos, os documentos escritos nesse serviço devem ser totalmente escritos Latex, exigindo que o usuário tenha total domínio da linguagem, ou disponha de tempo para pesquisar a utilização de recursos muitas vezes simples como a adição de imagens aos documentos. A curva de aprendizagem para escrita no editor é o principal ponto negativo para utilização do software.

Figura 14 – Editores de texto: Sharelatex




Fonte: https://goo.gl/YYAsgY

#### 2.3.3.5 FAST FORMAT

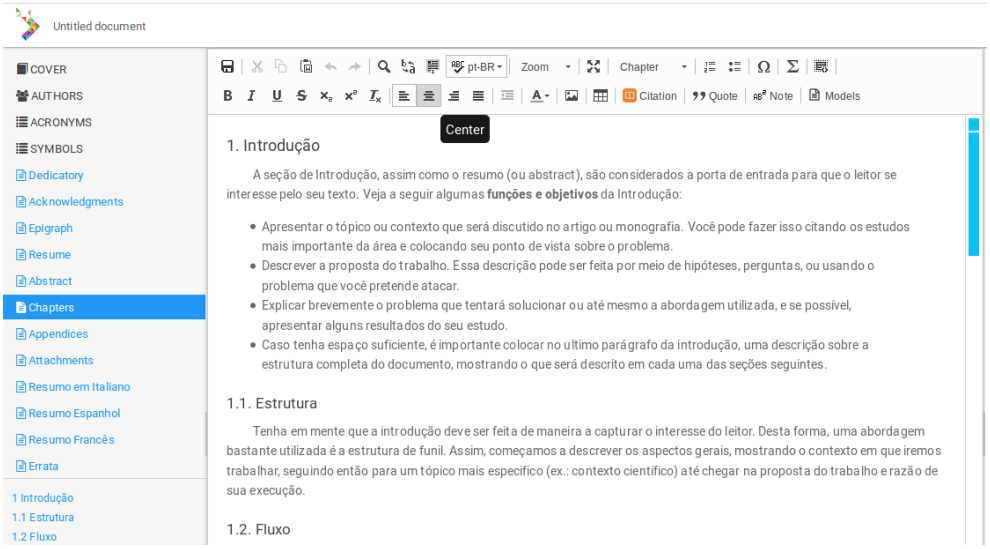
Existe ainda no mercado um software que é distribuído e comercializado no formato SaaS (Software as a service), que tenta unir a facilidade das boas aplicações de formatação gráficas de desktop e a qualidade que a linguagem Latex proporciona: o fastformat.

Este software trabalha na nuvem, apresenta uma boa interface gráfica para a formatação de trabalho e compila o conteúdo em Latex, ou seja, a tentativa é fazer a união dos dois mundos: a facilidade de uma boa interface gráfica com a qualidade da formatação do Latex.

A grande limitação do fastformat talvez seja a maneira que é feita a cobrança para a utilização. Enquanto em softwares pagos de desktop comumente temos o pagamento de uma licença para utilização da aplicação, aqui temos uma cobrança feita por tempo de utilização e por quantidade de espaço disponível para o documento. Ou

seja o usuário precisa ter a disciplina para finalizar seu trabalho no tempo estabelecido e também limitar o tamanho de seu arquivo de acordo com o que foi estabelecido na compra, caso contrário poderá ter problemas para gerar seu documento.

Figura 15 – Editores de texto: Fast Format




Fonte: *PrintScreen* dos autores do site fastformat.co

### 2.3.4 RESUMO

As opções apresentadas ao longo desse capítulo são vastas - pelo menos uma dezena de softwares, entre os mais tradicionais e utilizados editores de textos até algumas opções menos populares. Mesmo assim, não foi possível aliar a experiência do usuário como uma interface intuitiva e formatação avançada, encontra-se ora uma coisa, ora outra.

Os softwares tradicionais se concentram mais na experiência do usuário, trazendo comandos que são claros o bastante. Softwares de edição especializados em Latex costumam pecar na facilidade de uso: escrever em Tex pode ser bem desafiador nas primeiras tentativas. Mesmo após adquirir alguma experiência, alguns usuários ainda podem achar desconfortável ter que parar a produção do conteúdo para escrever códigos e tags de formatação - e ainda por cima ter que compilar o documento para visualizar as alterações estéticas.

Dentre os softwares e modelos analisados, os considerados mais adequados são os disponibilizados online - sem necessidade de configurar localmente o ambiente Tex nem nenhum tipo de instalação, possibilidade de compartilhar os documentos online e rápida adesão pelo público que os utiliza. Certamente a intenção do AbnTexTor não é ser mais um software de edição de texto, mas a junção do melhor de cada modelo para que o usuário tenha uma ferramenta que facilite a produção de textos científicos,

esse é sim o diferencial a ser buscado.

# 3 MATERIAIS E MÉTODOS

Este capítulo descreve todo o processo de desenvolvimento da aplicação. Aborda-se a utilização da metodologia ágil, sua adaptação às necessidades da equipe, o cronograma de atividades e os artefatos produzidos ao longo do desenvolvimento.

## 3.1 DESENVOLVIMENTO ÁGIL

Metodologias para desenvolvimento ágil de software começaram a surgir na década de 1990 como resposta às necessidades de agilizar a entrega de softwares, quebrando paradigma da modelagem estruturada de sistemas e a produção antecipada de todos os artefatos e modelagens, que tornavam o processo mais demorado. O documento que inaugurou essa tendência foi o Manifesto Ágil, assinado em 2001 e que continha as seguintes premissas (BECK et al., 2014):

- Valorizar indivíduos e suas interação mais do que processos e ferramentas
- Valorizar software em funcionamento mais do que documentação abrangente
- Colaboração com o cliente mais do que negociação do contrato
- Responder às mudanças mais do que seguir um plano

Essas premissas são opostas ao desenvolvimento tradicional de software, que tenta seguir um plano traçado antes do inicio do desenvolvimento do software, que produz uma série de artefatos e documentos sobre o sistema, que está atrelado à ferramentas e processos engessados e que não tem uma relação de proximidade com o cliente.

Um dos principais expoentes da metodologia de desenvolvimento ágil é o *framework SCRUM*, que tem como principais características a adoção de valores, princípios e práticas que podem ser adaptados às necessidades da organização (VIREIRA, 2014). A base do *SCRUM* está fundamentada em papéis, atividades e artefatos. Os papéis fundamentais são:

- *Product Owner*: é o responsável pelo produto, quais funcionalidades serão desenvolvidas e em qual ordem. Deve estar em constante contato com o time de desenvolvimento e com o *Scrum master*, sendo ponto central do processo de comunicação. Para Vieira (2014), é o papel responsável pelo "sucesso global da solução".

- *Scrum Master*: é o responsável por manter os valores, princípios e práticas do *SCRUM* entre os membros envolvidos. Tem também um papel de facilitador na resolução de problemas, previnindo interferências externas e removendo barreiras que podem comprometer a produtividade.
- Time *scrum*: o time de desenvolvimento *SCRUM* é auto-organizado e multidisciplinar. As pessoas da equipe de desenvolvimento são responsáveis pela concepção, construção e testes do produto. São equipes pequenas eque devem possuir todas as habilidades necessárias para produzir, com qualidade, o software desenvolvido (VIREIRA, 2014).

Algumas das atividades e artefatos do *framework* são:

- *Product Backlog*: é o documento que contém o que será desenvolvido e qual a prioridade dos itens previstos. É um documento que está em constante evolução e pode ser alterado à medida em que o software é construído.
- *Sprint*: são ciclos de trabalhos de duração fixa em que as funcionalidades definidas no *product backlog*. Geralmente tem duração fixa, não ultrapassando um mês.
- *Sprint Planning*: antes da execução do *Sprint* propriamente dito, o *Product Owner*, o time de desenvolviment e o *Scrum Master* definem os objetivos do próximo ciclo ou iteração e quais itens do *Product Backlog* serão implementados.
- *Daily Scrum*: também chamada de *Stend-up meeting*, é uma reunião de curta duração (até 15 minutos) feita diariamente entre os membros da equipe de desenvolvimento para alinhar as atividades desenvolvidas ao objetivo da iteração e compartilhar dificuldades e impedimentos para sua realização.
- *Sprint Review, Sprint Retrospective*: realizadas após a execução de um *Sprint*, tem como objetivo apresentar as funcionalidades desenvolvidas e verificar a necessidade de adaptações no produto ou no processo de trabalho.

Cohn elenca seis motivos para que equipes de desenvolvimento passem a usar a metodologia SCRUM (COHN, 2011, p. 33):

- Maior produtividade e menores custos
- Maior engajamento e satisfação no trabalho por parte dos funcionários
- Time-to-market mais rápido

- Maior qualidade
- Maior satisfação dos *stakeholders*
- O que estamos fazendo não funciona mais

Apesar de reconhecer a dificuldade de mensurar a produtividade de programadores, o autor trás dados de pesquisas que mostram aumento de produtividade na adoção de métodos ágeis, tendo como consequência uma redução dos custos: "Segundo a lógica, se as pessoas forem produtivas, os custos serão menores"(COHN, 2011, p. 34). Um dos fatores para o aumento de produtividade, segundo o autor, está diretamente ligado à maior satisfação dos funcionários com o trabalho que estão realizando, aliado ao fato de que com a adoção da metodologia ágil eles passaram a fazer menos horas-extras (COHN, 2011, p. 36).

Outra vantagem é que diferente dos métodos tradicionais, onde todo o planejamento é realizado antes do inicio da construção do software, métodos ágeis "tentem a lançar produtos mais rapidamente"(COHN, 2011, p. 36). Isso se deve a dois fatores: a maior produtividade da equipe leva a um desenvolvimento de funcionalidades mais rápido e as equipes ágeis costumam entregar versões incrementais do software.

Para Cohn, a maior qualidade da metodologia ágil está relacionada ao ritmo de trabalho das equipes ágeis, que costumam produzir consistentemente trabalho de maior qualidade, evitando que erros produzidos em algum momento do desenvolvimento voltem a ser tornar um problema em momentos posteriores. Além disso, metodologias ágeis empregam recursos como refatoração, programação em pares e testes automatizados.

Com os elementos citados acima, Cohn acha natural que os demais envolvidos no processo também apresentem maior satisfação, alertando para o fato da metodologia ágil ainda possuir um outro diferencial: a metodologia ágil permite que sejam feitas alterações de requisitos e prioridades (COHN, 2011, p. 38).

Por fim, o autor reconhece a inadequação de metodologias de desenvolvimento tradicionais, alertando para o fato de que "quando um processo que funcionava no passado deixa de funcionar, uma tendência comum é que ele continue sendo executado"(COHN, 2011, p. 39).

## 3.2 ADAPTAÇÃO DA METODOLOGIA

Para a adoção de uma metodologia ágil para o desenvolvimento do software proposto nesse trabalho são necessárias algumas adaptações. Primeiramente pela composição da equipe de apenas dois integrantes: os papeis de *Product Owner*, *Scrum*

*master* e time de desenvolvimento não puderam ser adequadamente distribuidos, sendo necessário que ambos os integrantes cuidassem do *Product backlog* e demais atividades.

Outra adaptação foi a não adoção da reunião diária. Ela foi substituída por uma reunião semanal, usualmente na segunda-feira, na qual eram discutidos quais seriam os objetivos para aquela *Sprint*, de acordo com o *Product backlog*, fixando sempre a sexta-feita subsequente como prazo final, dia em que eram discutidas as principais dificuldades enfrentadas, quais itens se encontravam atrasados e o quais ações seriam executadas no fim de semana para a adequação aos prazos.

## 3.3 CRONOGRAMA DE ATIVIDADES

Tabela 1 – Cronograma de atividades

| Iteração | Data final | Atividades |
|---|---|---|
| Sprint 0 | 11/08 | Definição de tecnologias utilizadas |
| | | Teste de conceito |
| Sprint 1 | 25/08 | Criação do ambiente de desenvolvimento virtualizado |
| | | Criação de repositórios Git |
| | | Banco de dados: DER |
| | | Teste de relacionamentos e conversão html para Latex |
| Sprint 2 | 08/09 | Implementação da gema via Gemfile |
| | | Rotas aninhadas |
| | | Correção do banco de dados |
| | | Pesquisa de sistemas para capítulo 2 |
| | | Iniciar capítulo 1 |
| Sprint 3 | 22/09 | Desenhar mockup de telas |
| | | Implementar materialcss para layout |
| | | Implementar build (tradução de trabalhos) |
| | | Implementar visualização de pdf |
| | | Inicio capítulo 2 (Mercado de editores de texto) |
| Sprint 4 | 06/10 | Implementação de reordenação para *Template* |
| | | *Section* e *Field* |
| | | Rotas amigáveis |
| | | Filtro de conteúdo nos Controllers |
| | | Expansão da gema Html2latex, elementos básicos |
| Sprint 5 | 13/10 | Sistema de login (permissões e acessos) |
| | | Correção no carregamento de JavaScript |
| | | Finalização do capítulo 1 |
| | | Finalização do capítulo 2 |

Tabela 1 – Cronograma de atividades

| Iteração | Data final | Atividades |
|---|---|---|
| | | Ajuste para regras de build para diferentes tipos de campos |
| Sprint 6 | 20/10 | Validações dos Models para os formulários |
| | | Sistema de paginação para Models |
| | | Diagrama de casos de uso |
| | | Diagrama de classes |
| | | Tabela de atividades |
| | | Início capítulo 3 |
| Sprint 7 | 27/10 | Especificações de caso de uso |
| | | Diagrama de sequencia |
| | | Cadastro e validação de usuário via formulário de cadastro |
| | | Cadastros de *templates* canônicos e testes |
| | | Início capítulo 4 |
| Sprint 8 | 03/11 | Ajustes no DER |
| | | Inclusão de avatar para usuário (Gravatar) |
| | | Finalização do capítulo 3 |
| | | Finalização do capítulo 4 |

Fonte: os autores (2017)

### 3.3.1 SPRINT 0

A primeira *Sprint* do trabalho visa verificar a aplicabilidade da ideia da aplicação converter um documento com marcação HTML para Latex, pois este é o ponto crítico da aplicação, sem a qual não é possível desenvolver o restante do trabalho.

#### 3.3.1.1 TESTE DE CONCEITO

Em primeiro lugar foi escolhida a linguagem e o *framework* para o desenvolvimento da aplicação - Ruby e Ruby on Rails. Em seguida, foi feita a construção inicial da *Gem* desenvolvida para a aplicação e que é responsável pela conversão da marcação da linguagem. Esses conceitos são explicados na Seção 3.3.1.2. Os testes iniciais foram bem sucedidos na medida em que o input em HTML resultava no output em Latex esperado

#### 3.3.1.2 TECNOLOGIAS UTILIZADAS

A escolha das tecnologias utilizadas é um ponto inicial de discussão, entendendo que essas escolham podem afetar de forma positiva ou negativa o que se

pretende desenvolver. A tabela apresenta as tecnologias utilizadas para o inicio do desenvolvimento do projeto, a versão utilizada e uma breve descrição de sua finalidade.

Tabela 2 – Tecnologias

| Tecnologia | Versão | Descrição/Finalidade |
|---|---|---|
| Ruby | 2.3.3 | Linguagem de programação orientada à objetos |
| Ruby on Rails | 5.0.1 | *Framework* web construído em Ruby |
| Gem | 2.5.2 | Gerenciador de pacotes da linguagem Ruby |
| MySQL | 14.14 | Banco de dados |
| Vagrant | 1.7.9 | Virtualização do ambiente de desenvolvimento |
| Git | 2.7.4 | Controle de versão |
| texlive | 3.14159265 | Ambiente Tex para compilação de arquivos Tex |

Fonte: os autores (2017)

Além dos itens selecionados pelos integrantes do grupo da Tabela 2, os ambientes de desenvolvimento usados para a construção do software são descritos a seguir.

#### 3.3.1.3 HARDWARE E SOFTWARE

- Computador 1: Notebook com processador Intel I5, 8Gb de memória RAM, rodando sistema operacional Linux Ubuntu Mate 16.04, kernel versão 4.8.0
- Computador 2: Computador de mesa com processador AMD FX- , 6Gb de memória RAM, rodando sistema operacional Linux Ubuntu (Zorin) 16.04.1, kernel 4.10.0
- Computador 3: Notebook com processador Intel I5, 8Gb de memória RAM, rodando sistema operacional Linux Ubuntu 16.04, kernel 4.10.0
- Ambiente Vagrant: sistema operacional Linux Ubuntu 16.04.3, kernel versão 4.4, com 1,5Gb de memória RAM
- Sharelatex: ambiente de edição compartilhada e compilação em núvem de documentos Latex
- Moqups: serviço de edição e compartilhamento online de prototipação
- Draw.io: serviço de criação online e compartilhamento de esquemas e diagramas

#### 3.3.1.4 GEMS, PLUGINS E *FRAMEWORKS*

O corpo desse Capítulo faz referência às *Gems* utilizadas. O sistema de *Gem* implementa no *framework* Ruby on Rails e ao ambiente da linguagem Ruby em geral o

uso de "pacotes"com funcionalidades extras desenvolvidos e mantidos por terceiros. A comunidade da linguagem é conhecida pela manutenção ativa do sistema de *Gems*. O repositório oficial de *Gems* é o https://rubygems.org/, que na data de escrita desse trabalho tem mais de 137 mil *Gems* hospedadas, mais de 116 mil usuários cadastrados e contabiliza mais de 17 bilhões de downloads. Além disso, também é possível utilizar *Gems* diretamente de repositórios do GitHub e também locais. A *Gem* criada para esse trabalho está hospedada no https://rubygems.org/. A lista completa de *Gems* utilizadas está na tabela a seguir.

Tabela 3 – Lista de *Gems*

| *Gem* | Versão | Finalidade |
|---|---|---|
| rails | 5.0.5 | *Framework* web MVC em Ruby |
| mysql2 | 0.3.18 | Conexão e manipulação do banco de dados |
| puma | 3.0 | Servidor da aplicação rails |
| sass-rails | 5.0 | *framework* CSS integrado ao sistema |
| uglifier | 1.3.0 | minificador de JavaScript |
| jquery-rails | 4.3.1 | *framework* JavaScript integrado ao sistema |
| tinymce-rails | n/a | Editor de texto WYSIWYG |
| html2latex | 1.0.0 | conversão de *tags* HTML para Latex |
| acts_as_list | 0.9.9 | reorganização dos itens baseados em sua posição |
| materialize-css | 0.100.2 | *framework* de estilos CSS |
| devise | 4.3.0 | autenticação |
| cancancan | 2.0 | autorização |
| will_paginate | 3.1.0 | paginação de tabelas |

Fonte: os autores (2017)

A versão da *Gem tynemce_rails* não foi especificada porque a equipe está utilizando um *fork* hospedado no *github* de um dos integrantes.

### 3.3.2 SPRINT 1

Essa *Sprint* visa criar a estrutura utilizada para o desenvolvimento. A criação de um ambiente de desenvolvimento virtualizado com Vagrant, contendo todas as tecnologias necessárias para o desenvolvimento: Ruby, Ruby on Rails, MySQL e o ambiente Latex para compilação de documentos. Também é nessa Sprint criado o repositório Git para o controle de versão, hospedados no serviço BitBucket (http://www.bitbucket.org). É criada a aplicação Ruby on Rails já no ambiente controlado no Vagrant. Com a aplicação rodando é possível testar os relacionamentos entre cada entidade do banco de dados criada no Diagrama Entidade Relacionamento.

#### 3.3.2.1 DER

Uma das etapas dessa iteração é o desenvolvimento do Diagrama Entidade Relacionamento (DER) para o banco de dados relacional da aplicação. O DER se encontra no Apêndice A. A modelagem de dados é constantemente revisada durante a construção da aplicação, de modo que esta é a versão inicial da mesma, que pode sofrer alterações, como apresentado na Seção 3.3.9. A modelagem do banco de dados cria 8 entidades: *Paper*, *Template*, *Section*, *Field*, *Content*, *Image*, *User* e *Reference*. Mais detalhes sobre cada uma delas e suas cardinalidades estão na lista abaixo:

- *Paper* representa um trabalho criado pelo usuário, este representado pela tabela *User*. *Paper* e *User* tem uma relação muitos-para-um.
- A entidade *Template* representa um modelo de trabalho disponível no sitema e o relacionamento de *Paper* com *Template* é de muitos-para-um.
- *Section* representa as diferentes partes de um *Template*, como "capa", "folha de rosto", "conteúdo"e assim por diante. A *Section* pode ou não ser editável ou usar o editor do sistema. *Section* e *Template* tem relacionamento muitos-para-um.
- *Field* representa um campo a ser preenchido dentre de uma *Section* e um *Field* pode ou não ser um alvo da conversão HTML - Latex de acordo com a *Section* em que se encontra.
- Um trabalho tem diferentes campos e cada campo pode ser usado em diferentes trabalhos. Logo, a relação entre *Paper* e *Field* é de muitos-para-muitos, gerando a entidade *Content*.
- A entidade *Image* representa uma imagem associada a um trabalho. Um trabalho pode ter várias imagens, então o relacionamento entre *Image* e *Paper* é de muitos-para-um. A entidade
- *Reference* representa uma referência bibliográfica, e se relaciona com *Paper* na forma de muitos-para-muitos, gerando uma tabela de ligação.
- Existe uma relação de muitos-para-muitos entre *Paper* e *User*, para o compartilhamento de trabalhos.

### 3.3.3 SPRINT 2

O objetivo dessa *Sprint* é trazer as primeiras melhorias para os produtos gerados nos testes de conceito: a *Gem* desenvolvida para a conversão de marcação HTML - Latex foi implementada na aplicação Ruby on Rails por meio do *Gemfile* [1]. A

[1] Arquivo presente em todas as aplicações Ruby on Rails, onde são especificadas todas as *Gems* utilizadas.

estrutura da aplicação criada na iteração anterior foi alterada para que as *Sections* fossem aninhadas aos *Templates* e os *Fields* às *Sections*.

Algumas correções pontuais foram feitas no banco de dados da aplicação para que ele se adequasse ao DER, uma vez que foram desenvolvidos na mesma iteração - em geral foram feitas algumas renomeações de campos, inclusão e exclusão de campos adicionais.

Os primeiros elementos do Capítulo 1 começam a ser escritos, com foco na introdução. Iniciam-se as pesquisas necessárias para a escrita do segundo capítulo desse trabalho, procurando outros sistemas e editores de texto para trabalhos acadêmicos.

### 3.3.4 SPRINT 3

Nessa *Sprint* é construído um *Controller* específico para a conversão HTML - Latex, chamado de *Build* do trabalho. Até então essa conversão era feita no mesmo *Controller* que atualizava o conteúdo do trabalho (*Contents*). Essa alteração visa aumentar a coesão do sistema.

Juntamente com essa alteração, é definido que o trabalho compilado em PDF deve ser acessado diretamente através da URL, sendo disponibilizado na pasta pública do *framework*. Como forma de proteção ao material exposto de tal forma, é definido que o usuário acessa seus trabalhos apenas utilizando um nome "público", não sendo relacionado ao *id* do trabalho no banco de dados. O nome público é feito a partir de uma *hash* produzida a partir da *id* do usuário, o *id* do trabalho e a hora do sistema, utilizando os oito primeiros algarismos alfanuméricos da sequência gerada.

#### 3.3.4.1 TELAS

Ainda nessa iteração são feitos os desenhos das telas. Para tal, utiliza-se o Moqups (https://moqups.com/), uma ferramenta online e gratuita onde é possível criar os mockups de telas, armazenar e compartilhar. Para a construção do layout é definida a utilização do *framework* Materializecss, implementado via *Gemfile*. As telas projetadas nessa etapa estão no Apêndice B.

### 3.3.5 SPRINT 4

Para essa *Sprint* são planejados elementos de usabilidades importantes para o funcionamento do sistema. Isso porque a conversão HTML - Latex planejada para o sistema depende da ordem em que os elementos são extraído do banco de dados. A ordenação de *Template*, *Section* e *Field* são implementados por meio de uma *Gem* específica para esse fim.

Para melhorar a usabilidade do sistema, são criadas rotas (URLs) amigáveis para o usuário, substituindo os padrões gerados em inglês para rotas em português para acesso aos trabalhos e para a edição do cadastro ("perfil") do usuário. Para o acesso de trabalhos (Paper), também são implementadas rotas que utilizam os nomes públicos dos trabalhos, assim o usuário não tem acesso ao *id* armazenado no banco de dados. Também são implementados filtros nos *Controllers* da aplicação, para que apenas os conteúdos referentes ao usuário sejam mostrados em tela.

Outro grande avanço planejado para essa etapa é a primeira expansão significativa da *Gem* Html2latex desde sua construção para o teste de conceito. São adicionadas as principais *tags* de marcação de texto, como negrito, itálico, títulos e subtítulos, codificação de acentos e alguns caracteres especiais.

### 3.3.6 SPRINT 5

Nessa *Sprint* são implementados os sistemas de login e permissões de acesso do aplicativo, por meio de *Gems* específicas para esse fim. Os usuários sem permissão de administrador têm acesso à sua página de perfil para alterar informações e troca de senha, à tela de cadastro e edição dos trabalhos e ao trabalho compilado em PDF. Os usuários com permissão de administrador podem, além das páginas acima, acessar o cadastro de *Template*, *Section*, *Field* e *User*, tendo acesso completo ao sistema.

Nessa iteração é implementada uma melhoria no carregamento do JavaScript das páginas, que causava a quebra dos seletores de texto do *framework* de CSS (Materialize), sendo necessário o recarregamento da página. Além disso, é feita um ajuste na função de iteração entre os diferentes campos do trabalho, adequando a chamada da tradução HTML - Latex apenas para as seções que possuam o editor de texto e a implementação da divisão de valores em campos multivalorados (cujos diferentes valores passam a ser divididos por vírgulas).

Em paralelo, é feita a finalização da escrita dos capítulos 1 e 2 do trabalho para posterior avaliação do professor orientador.

### 3.3.7 SPRINT 6

Nesta *Sprint* são implementadas as validações em diversos formulários da aplicação, sendo necessário para esse fim a aplicação de regras de validação e mensagens de erros nos *Models* correspondentes. As regras de validação implementadas são:

- *Template*: o campo *name* deve ser único e o campo *description* é obrigatório;

- *Section*: o campo *name* é obrigatório e único para um *Template*, os campos *template_id* e *position* são obrigatórios;
- *Field*: o campo *name* é obrigatório e único para uma *Section*, os campos *open_tag*, *close_tag*, *label*, *position* e *section_id* são obrigatórios;
- *Paper*: o campo *name* é obrigatório e único para um *User*, os campos *user_id* e *template_id* são obrigatórios.

Além das regras de validação de formulários, é implementado um sistema de paginação, por meio de uma *Gem*, nas páginas que listam usuários, trabalhos, *templates*, seções e campos.

Para a monografia, é iniciado o Capítulo 3 com a tabela de atividades de todas as *Sprints* planejadas e executadas. São desenvolvidos os diagramas de caso de uso e de classes, conforme os item a seguir.

#### 3.3.7.1 DIAGRAMA DE CASOS DE USO

Para (BOOCH, 2012), os casos de uso podem ser aplicados para especificar qual será o comportamento do programa que está sendo modelado, sem se preocupar ainda com a implementação propriamente dita. Para os autores (2017) "Casos de uso bem estruturados denotam somente comportamento essencial do sistema ou subsistema e não são amplamente gerais, nem muito específicos"(BOOCH, 2012, p. 246).

O Diagrama de Casos de Uso está no Apêndice C. Ele apresenta a interação de dois atores com o sistema, sendo que o *Usuário* está restrito ao cadastro e edição de trabalhos (Paper e *Content*) e conversão (*build*) do trabalho e apresentação em PDF. O ator *Admin* tem acesso ao cadastro de *Template*, *Section* e *Field*.

#### 3.3.7.2 DIAGRAMA DE CLASSES

Os diagramas de classes apresentam um conjunto de classes e interfaces e como esses elementos interagem entre si. Para Booch, os diagramas de classes são importantes para visualização, especificação e documentação do sistema (BOOCH, 2012, p. 115).

O diagrama de classes apresenta basicamente a estrutura dos modelos (*Models*) da aplicação. A estrutura "MVC"será explicada no capítulo 4, juntamente com a arquitetura do sistema. A grosso modo, os modelos representam os objetos efetivamente modelados para o funcionamento do sistema. Temos um conjunto de 6 modelos: *Template*, *Section*, *Field*, *User*, *Paper* e *Content*. A relação entre eles é análoga ao que foi exposto sobre o DER do sistema, um *Paper* contém um objeto *User* e um objeto

*Template*; este contém uma coleção de objetos do tipo *Section*, que contém sua própria coleção de objetos *Field*. Cada pedaço de conteúdo do trabalho (Paper) é representado pela classe *Content*, que contém um objeto *Paper* e um objeto *Field*.

O diagrama de classes está no Apêndice D.

### 3.3.8 SPRINT 7

Essa iteração produz dois artefatos da modelagem do sistema, os diagramas de sequência e as especificações de caso de uso, tratados abaixo. Nessa semana também é iniciada a escrita do Capítulo 4, com a apresentação do sistema.

Em relação ao desenvolvimento da aplicação, é implementado um cadastro de usuário mais completo. Isso porque a *gem* utilizada para autenticação cria um formulário básico com apenas e-mail e senha. Para que o usuário não opere o sistema sem informações básicas, como nome e nome de usuário, o formulário é adaptado, o *controller* também é alterado para receber os novos parâmetros da requisição e as devidas validações são implantadas no *model*: nome completo não pode ser vazio, o nome de usuário e e-mail não podem ser vazios e devem ser únicos, e a senha deve ser preenchida e repetida com o mesmo conteúdo no campo de confirmação de senha.

Essa iteração deu inicio ao cadastro de *Templates* para cadastro de trabalhos, seguindo as diretrizes da ABNT.

#### 3.3.8.1 DIAGRAMA DE SEQUÊNCIA

Os diagramas de sequência são diagramas que tratam da comunicação entre os elementos do sistema, dando ênfase à ordem em que as mensagens são transmitidas entre eles (BOOCH, 2012, p. 273). Os diagramas de sequência são um dos últimos artefatos documentais produzidos para o trabalho, tendo visto que "para a elaboração de um diagrama de sequência, é importante que os objetivos do sistema já tenham sido levantados [...]"(DEBONI, 2003, p. 118). Os diagramas de sequência estão no apêndice E e são descritos com mais detalhes nas especificações caso de uso.

#### 3.3.8.2 ESPECIFICAÇÕES DE CASO DE USO

As especificações de caso de uso tratam da ordem dos acontecimentos das interações entre o sistema e o usuário para o caso de uso que está sendo analisado. As especificações elaboradas são referentes aos diagramas de sequência e podem ser encontrados no Apêndice F.

### 3.3.9 SPRINT 8

A última *Sprint* é dedicada à finalização do trabalho e ajustes necessários à documentação, principalmente ao DER elaborado no início do trabalho, uma vez que nem todas as funcionalidades modeladas para o banco de dados foram desenvolvidas para a versão entregue.

O ajuste ao DER retira as tabelas *Image* e *References*, tendo em vista que representam um incremento ao escopo inicial desenhado para o projeto, que a decidiu-e não executar. Como consequência direta, a tabela que relaciona *Reference* e *Paper* também sai do diagrama, bem como a tabela da ligação muitos-para-muitos entre *Paper* e *User*. O novo DER está no Apêndice G.

A funcionalidade desenvolvida para o sistema nessa iteração é o uso do serviço *Gravatar*[2] para mostrar imagens do usuário (*avatar*). Para o trabalho escrito, são concluídos o capítulo 3, com ajuste das citações e bibliografia, e o Capítulo 4 com a apresentação do sistema.

### 3.3.10 FASE DE CONCLUSÃO

A fase de conclusão do trabalho teve início no dia 06/11. Nessa fase, o desenvolvimento de novas funcionalidades foi interrompido e foi dado foco no aprimoramento de funcionalidades já existentes. As funcionalidades previstas que deixaram de ser desenvolvidas fazem parte do escopo do capítulo 5. O aprimoramento do programa passou pelo melhoramento de elementos visuais, tratamento de erros e exceções e refinamento da aplicação de regras de negócio.

O balanço dos pontos positivos e negativos, dificuldades técnicas e conceituais também foram debatidos para integrarem as considerações finais. Foi elaborada uma lista de melhorias e mudanças para a continuação do desenvolvimento, incluindo itens como design, funcionalidades, modelagem conceitual, entre outros. Espera-se que a disponibilização do código fonte traga novos elementos e ideias para o futuro da aplicação.

A preparação para a apresentação do software produzido passou pelas etapas de empacotamento da versão final em ambiente de virtualização, para que o ambiente de execução estivesse o mais controlado possível. Os modelos e trabalhos elaborados dentro do sistema para apresentação à banca avaliadora também foram exportados para a carga inicial do banco de dados (*seed*).

[2] https://gravatar.com/

# 4 APRESENTAÇÃO DO SISTEMA

Neste capítulo é apresentado o funcionamento da aplicação. Inicialmente mostra-se mais a fundo como o *framework* escolhido trabalha para a aplicação. Também são vistas as possibilidades que os usuários (comuns e administradores) tem de interação com o sistema, manipulando *templates*, criando, editando e gerando documentos e outras funcionalidades. Outro ponto mostrado é um detalhamento maior do ciclo de vida de um documento, explica-se: desde a criação de seu *template* até a geração do documento final no formato PDF.

## 4.1 ARQUITETURA DO FRAMEWORK

A arquitetura de sistema do AbnTexTor é essencialmente baseada em tecnologia web, ou seja existe uma linguagem de programação por trás controlando a lógica enquanto a apresentação é feita em HTML que é renderizado em qualquer navegador atual. O ponto que evita que toda a aplicação seja considerada 100% web é que existe uma chamada ao sistema para o software *texlive* que faz a compilação do documento final.

O sistema do AbnTexTor é divido em pelo menos 3 camadas:

- **Front End**: é a camada de apresentação ao usuário. Por se tratar de uma aplicação web, são as páginas HTML que a compõe.
- **Back End**: é a camada onde é feita toda a lógica de programação pesada gerenciando usuários, documentos e etc.
- **Chamada ao sistema**: essa última camada compõe uma chamada a uma aplicação terceira do sistema operacional para geração e compilação do documento Tex.

Quem gerencia essas três camadas é um *framework* da linguagem Ruby que utiliza o padrão MVC em sua arquitetura: o Ruby on Rails.

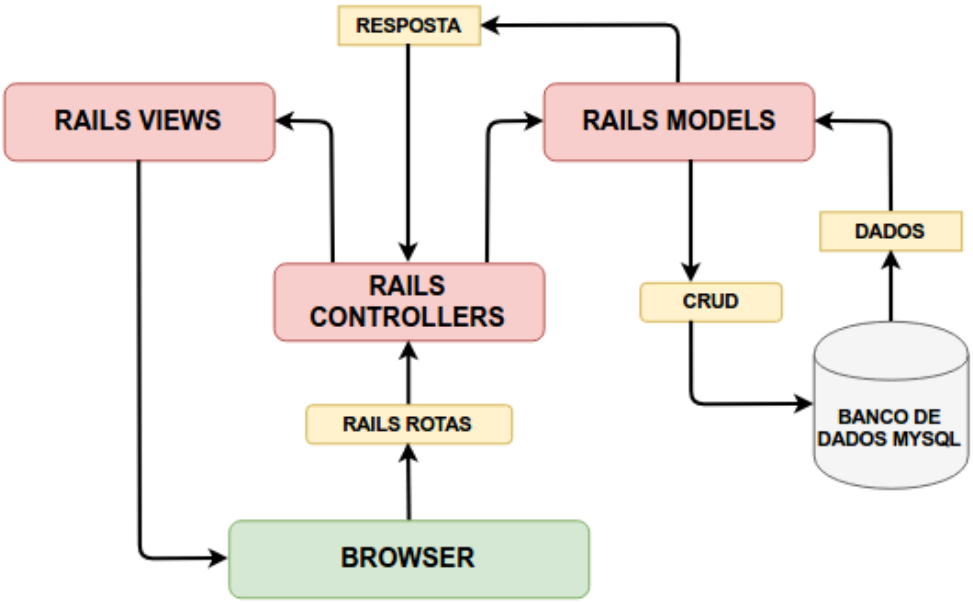
### 4.1.1 RUBY ON RAILS E MVC

Ruby on Rails é um famoso e poderoso *framework* MVC utilizado na linguagem Ruby, toda a funcionalidade do modelo MVC (Model, Views, Controller) é de fácil manipulação no *framework* gerando assim aplicações com códigos limpos e bem divididos. Ainda, sabe-se que o "O Rails também é usado por muitos desenvolvedores

já há bastante tempo, portanto já foi testado em diversas situações, como alta carga, ou grande número de usuários."(FUENTES, 2012, p. 74)

A estrutura básica da aplicação é mostrada na Figura 16, observa-se que é seguida a risca a arquitetura MVC graças a maneira que o Rails orquestra a aplicação.

Figura 16 – Apresentação do sistema: Model-View-Controller




Fonte: os autores (2017)

Seguindo o padrão MVC, então, tem-se o início da execução pelo *browser*, onde o cliente faz a requisição do serviço, assim que essa requisição é feita o arquivo de direciona para o *controller* executar o método correto para tratar o pedido do cliente, o *controller* envia uma requisição ao *model* que vai utilizar as regras de negócio para acessar o banco de dados que irá então enviar os dados pedidos de volta a esse *model*, que então responde ao *controller*, e por fim o *controller* chama uma *view* para renderizar no *browser* do cliente a informação pedido pela sua requisição inicial.

### 4.1.2 ROTAS

O arquivo que trata as rotas no *Ruby on Rails* é o /config/routes.rb, é nele que estão configuradas como as rotas devem ser construídas na aplicação de modo que ela saiba para quem direcionar a requisição assim que o usuário acessa a página escolhida no novegador. A Tabela 4 lista todas as rotas e seus métodos dentro da aplicação.

Tabela 4 – Rotas

| # | Rota | Controller | Requisição |
|---|---|---|---|
| 1 | /users/sign_in(.:format) | devise/sessions#new | GET |

Tabela 4 – Rotas

| # | Rota | Controller | Requisição |
|---|---|---|---|
| 2 | /users/sign_in(.:format) | devise/sessions#create | POST |
| 3 | /users/sign_out(.:format) | devise/sessions#destroy | DELETE |
| 4 | /users/password/new(.:format) | devise/passwords#new | GET |
| 5 | /users/password/edit(.:format) | devise/passwords#edit | GET |
| 6 | /users/password(.:format) | devise/passwords#update | PATCH |
| 7 | /users/password(.:format) | devise/passwords#update | PUT |
| 8 | /users/password(.:format) | devise/passwords#create | POST |
| 9 | /users/cancel(.:format) | devise/registrations#cancel | GET |
| 10 | /users/sign_up(.:format) | devise/registrations#new | GET |
| 11 | /users/edit(.:format) | devise/registrations#edit | GET |
| 12 | /users(.:format) | devise/registrations#update | PATCH |
| 13 | /users(.:format) | devise/registrations#update | PUT |
| 14 | /users(.:format) | devise/registrations#destroy | DELETE |
| 15 | /users(.:format) | devise/registrations#create | POST |
| 16 | / | *papers*#index | GET |
| 17 | /build/build(.:format) | build#build | GET |
| 18 | /trabalhos(/:page)(.:format) | *papers*#index | GET |
| 19 | /trabalhos(.:format) | *papers*#index | GET |
| 20 | /trabalho/:hash_name(.:format) | *papers*#edit | GET |
| 21 | /trabalho/:hash_name(.:format) | *papers*#destroy | DELETE |
| 22 | /trabalho/:hash_name/:section_id(.:format) | *contents*#edit | GET |
| 23 | /pdf/:hash_name(.:format) | build#build | GET |
| 24 | /perfil(.:format) | *users*#profile | GET |
| 25 | /contents/edit(.:format) | *contents*#edit | GET |
| 26 | /contents/update(.:format) | *contents*#update | POST |
| 27 | /papers(.:format) | *papers*#index | GET |
| 28 | /papers(.:format) | *papers*#create | POST |
| 29 | /papers/new(.:format) | *papers*#new | GET |
| 30 | /papers/:id/edit(.:format) | *papers*#edit | GET |
| 31 | /papers/:id(.:format) | *papers*#show | GET |
| 32 | /papers/:id(.:format) | *papers*#update | PATCH |
| 33 | /papers/:id(.:format) | *papers*#update | PUT |
| 34 | /papers/:id(.:format) | *papers*#destroy | DELETE |
| 35 | /templates/:template_id/sections(.:format) | *sections*#index | GET |
| 36 | /templates/:template_id/sections(.:format) | *sections*#create | POST |
| 37 | /templates/:template_id/sections/new(.:format) | *sections*#new | GET |
| 38 | /templates/:template_id/sections/:id/edit(.:format) | *sections*#edit | GET |
| 39 | /templates/:template_id/sections/:id(.:format) | *sections*#show | GET |
| 40 | /templates/:template_id/sections/:id(.:format) | *sections*#update | PATCH |
| 41 | /templates/:template_id/sections/:id(.:format) | *sections*#update | PUT |
| 42 | /templates/:template_id/sections/:id(.:format) | *sections*#destroy | DELETE |
| 43 | /templates(.:format) | *templates*#index | GET |
| 44 | /templates(.:format) | *templates*#create | POST |
| 45 | /templates/new(.:format) | *templates*#new | GET |
| 46 | /templates/:id/edit(.:format) | *templates*#edit | GET |

Tabela 4 – Rotas

| # | Rota | Controller | Requisição |
|---|---|---|---|
| 47 | /templates/:id(.:format) | *templates*#show | GET |
| 48 | /templates/:id(.:format) | *templates*#update | PATCH |
| 49 | /templates/:id(.:format) | *templates*#update | PUT |
| 50 | /templates/:id(.:format) | *templates*#destroy | DELETE |
| 51 | /sections/:section_id/fields(.:format) | *fields*#index | GET |
| 52 | /sections/:section_id/fields(.:format) | *fields*#create | POST |
| 53 | /sections/:section_id/fields/new(.:format) | *fields*#new | GET |
| 54 | /sections/:section_id/fields/:id/edit(.:format) | *fields*#edit | GET |
| 55 | /sections/:section_id/fields/:id(.:format) | *fields*#show | GET |
| 56 | /sections/:section_id/fields/:id(.:format) | *fields*#update | PATCH |
| 57 | /sections/:section_id/fields/:id(.:format) | *fields*#update | PUT |
| 58 | /sections/:section_id/fields/:id(.:format) | *fields*#destroy | DELETE |
| 59 | /sections(.:format) | *sections*#index | GET |
| 60 | /sections(.:format) | *sections*#create | POST |
| 61 | /sections/new(.:format) | *sections*#new | GET |
| 62 | /sections/:id/edit(.:format) | *sections*#edit | GET |
| 63 | /sections/:id(.:format) | *sections*#show | GET |
| 64 | /sections/:id(.:format) | *sections*#update | PATCH |
| 65 | /sections/:id(.:format) | *sections*#update | PUT |
| 66 | /sections/:id(.:format) | *sections*#destroy | DELETE |
| 67 | /users(.:format) | *users*#index | GET |
| 68 | /users(.:format) | *users*#create | POST |
| 69 | /users/new(.:format) | *users*#new | GET |
| 70 | /users/:id/edit(.:format) | *users*#edit | GET |
| 71 | /users/:id(.:format) | *users*#show | GET |
| 72 | /users/:id(.:format) | *users*#update | PATCH |
| 73 | /users/:id(.:format) | *users*#update | PUT |
| 74 | /users/:id(.:format) | *users*#destroy | DELETE |

Fonte: os autores (2017)

### 4.1.3 MODELS

Na criação de uma aplicação *Ruby On Rails*, é usado um componente chamado *ActiveRecord*. Esse componente é do tipo *Object Relational Mapping*, ou seja, ele mapeia as estruturas do banco de dados em objetos *Ruby*. Com isso é possível transformar chamadas a métodos *Ruby* em consultas *SQL* e depois mapear novamente em objetos *Ruby*, como no exemplo a seguir:

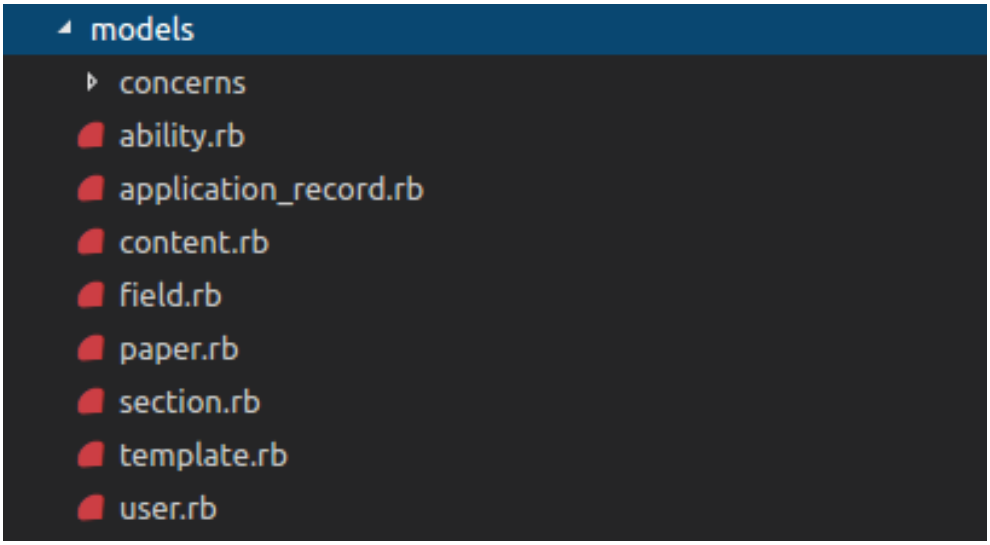
```
2.3.3 :005 > User.all
  User Load (0.6ms)  SELECT `users`.* FROM `users`
 => #<ActiveRecord::Relation [#<User id: 1, name: "Admin",
 alias: "admin", college: nil, is_active: true, is_admin: true,
 created_at: "2017-11-08 00:02:50", updated_at: "2017-11-11 12:55:37",
```

```
 email: "admin@admin.com">, #<User id: 2,
 name: "Fernando Valle", alias: "fernando", college: "ufpr",
 is_active: true, is_admin: false, created_at: "2017-11-08 02:52:07",
 updated_at: "2017-11-08 03:11:47", email: "phervalle@gmail.com">]>
2.3.3 :006 > User.where(:id => 2)
  User Load (0.7ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 2
 => #<ActiveRecord::Relation [#<User id: 2,
 name: "Fernando Valle", alias: "fernando",
 college: "ufpr", is_active: true, is_admin: false,
 created_at: "2017-11-08 02:52:07",
 updated_at: "2017-11-08 03:11:47", email: "phervalle@gmail.com">]>
```

Os *models* feitos com o *ActiveRecord* tem ainda várias caraterísticas como validação de atribudos, *callbacks*, traduções, dentre outras.

No AbnTexTor os *models* encontram-se no diretório /app/models/.

Figura 17 – Models utilizados no AbnTexTor




Fonte: os autores (2017)

### 4.1.4 VIEWS

No *Ruby On Rails* a biblioteca *ActionView* é quem auxilia na apresentação das páginas HTML para o usuário. Por padrão os *templates* podem ser montados usando o ERB (*Embedded Ruby*), bastando para isso ter a extensão .erb. Para misturar elementos HTML com o ERB, o arquivo deverá seguir o padrão de nome *nomearquivo.html.erb*. Códigos *Ruby* podem ser inseridos em arquivos ERB bastando estar entre as tags <% %> ou <%= %>.

Um exemplo de uma *view* da aplicação na Figura 18, onde é mostrada a *view* que lista os trabalho do usuário, é feito um *foreach* que lista todos os trabalhos do

usuário:

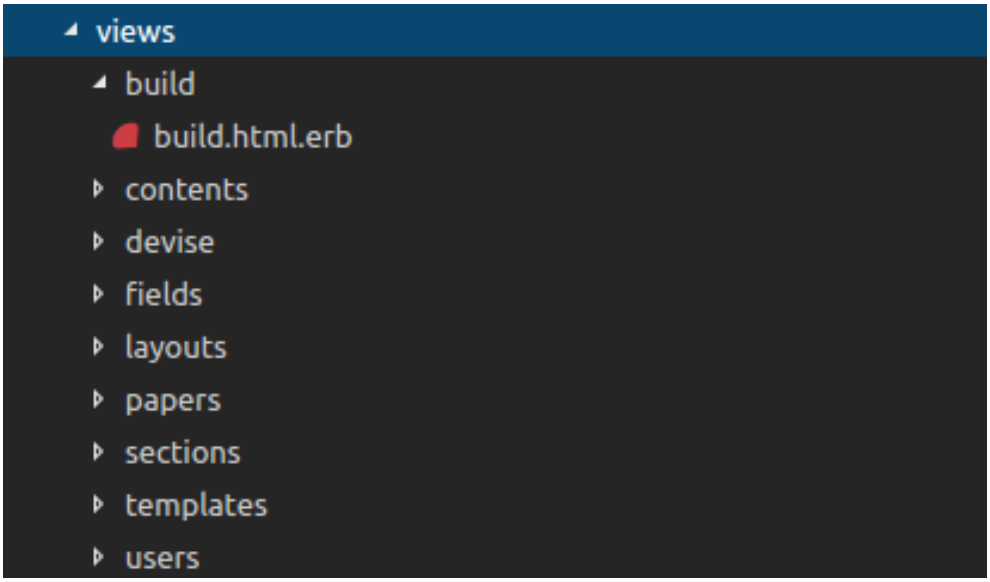
Figura 18 – View /app/views/papers/index.html.erb

```
<% @papers.each do |paper| %>
  <tr>
    <td><%= paper.id %></td>
    <td><%= link_to "#{paper.name}", "/trabalho/#{paper.hash_name}/#{paper.template.initial_section.id}" %></td>
    <td><%= paper.hash_name %></td>
    <td><%= paper.template.name %></td>
    <td><%= paper.created_at.strftime("%d/%m/%Y %H:%M:%S") %></td>
    <td><%= paper.updated_at.strftime("%d/%m/%Y %H:%M:%S") %></td>
    <td><%= link_to "<i class='material-icons'>picture_as_pdf</i>".html_safe, "/pdf/#{paper.hash_name}", class: "btn-floating" %></td>
    <td><%= link_to "<i class='material-icons'>edit</i>".html_safe, "/trabalho/#{paper.hash_name}", class: "btn-floating" %></td>
    <td><%= link_to "<i class='material-icons'>delete</i>".html_safe, "/trabalho/#{paper.hash_name}", method: :delete, data: { confirm: 'Excluir tra
  </tr>
<% end %>
```

Fonte: os autores (2017)

Na Figura 19 o diretório das *views*, que estão organizadas dentro de seus respectivos diretórios, encontram-se em /app/views/:

Figura 19 – Views utilizadas no AbnTexTor




Fonte: os autores (2017)

### 4.1.5 CONTROLLER

O componente principal que trabalha na camada de controle é o *ActionController*, é ele quem da suporte para escrever código para chamar os *models*, tratar requisições ou definir algumas regras na programação da aplicação.

Um exemplo de um trecho do *controller papers*_controller.rb, que é quem controla os trabalhos do usuário. Na Figura 20, é definido o método *index* como o principal do *controller*, então quando o usuário chamar esse *controller* na *url* (que pela tabela de rotas é a rota padrão da aplicação, ou seja, o *controller* é chamado pela raiz do *app*) será chamada o *model papers*, que irá retornar os registros contendo os trabalhos do usuário.

Figura 20 – Trecho do código do controller *papers*_controller.rb

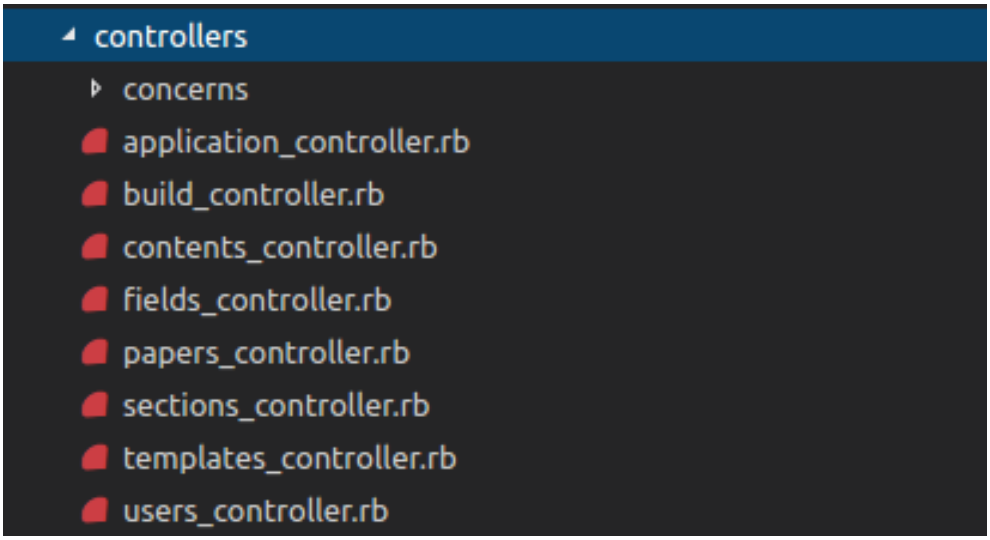
```
class PapersController < ApplicationController
  authorize_resource
  before_action :set_paper, only: [:show, :update]
  before_action :set_templates, only: [:new, :create, :edit]

  # GET /papers
  # GET /papers.json
  def index
    @papers = Paper.from_user(current_user.id).paginate(:page => params[:page], :per_page => 10)
  end
```

Fonte: os autores (2017)

Na Figura 21 é mostrada a lista dos controles usados no AbnTexTor.

Figura 21 – Controllers econtrados em /app/controllers/




Fonte: os autores (2017)

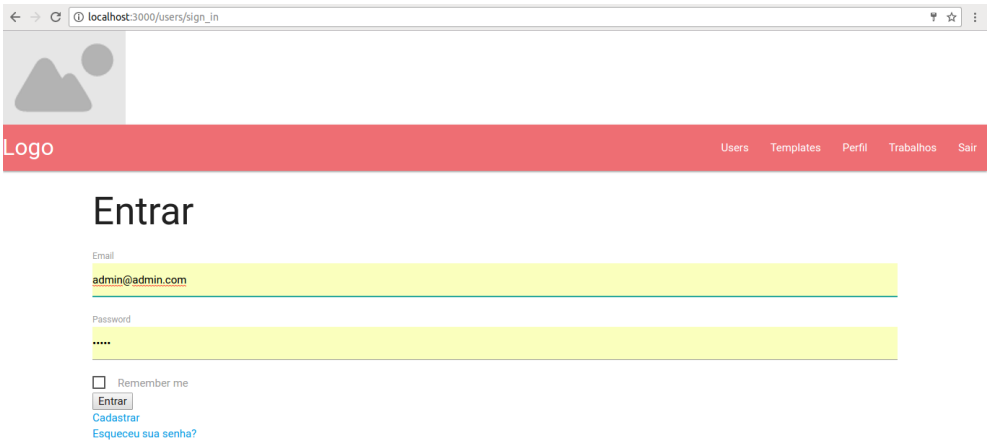
## 4.2 ARQUITETURA DO ABNTEXTOR

O AbnTexTor conta em seu sistema, telas e ações intuitivas para facilitar a interação dos usuários. A divisão de acesso é feita para dois atores: o usuário e o administrador. Como os nomes propõe, a ação do usuário é basicamente escolher o *template* que melhor sirva para seu projeto e editar e gerenciar seus documentos. O administrador, além de poder fazer isso, também pode criar e gerenciar *templates* e usuários.

### 4.2.1 USUÁRIO

Ao iniciar o AbnTexTor, Figura 22, é chamado o método *index* do *controller papers*_controller.rb (definido em /config/routes.rb). Porém, caso não exista usuário logado, a tela de *login* é enviada para o usuário se autenticar.

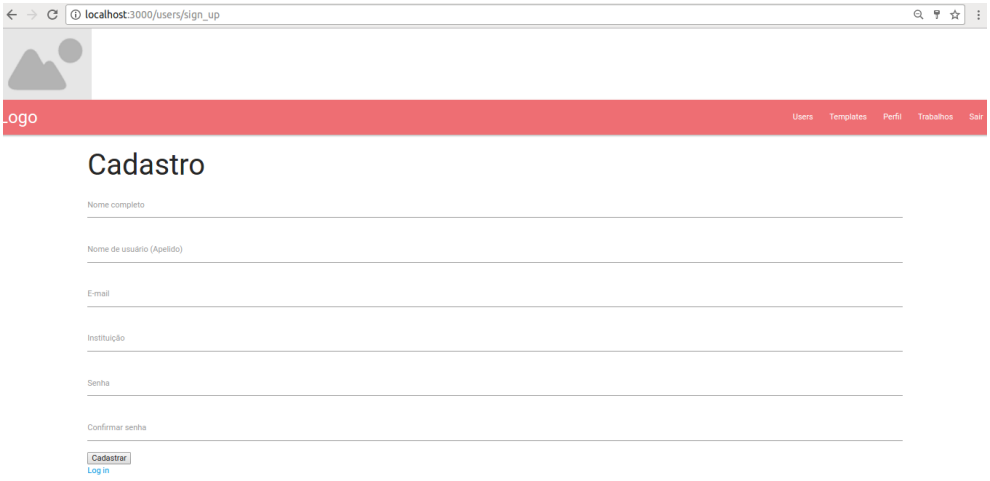
Figura 22 – Sistema: tela de login




Fonte: os autores (2017)

Ainda, caso o usuário não tenha acesso ao sistema, ele pode clicar na opção Cadastrar e fazer um rápido registro para acessar o sistema, como mostra a Figura 23.

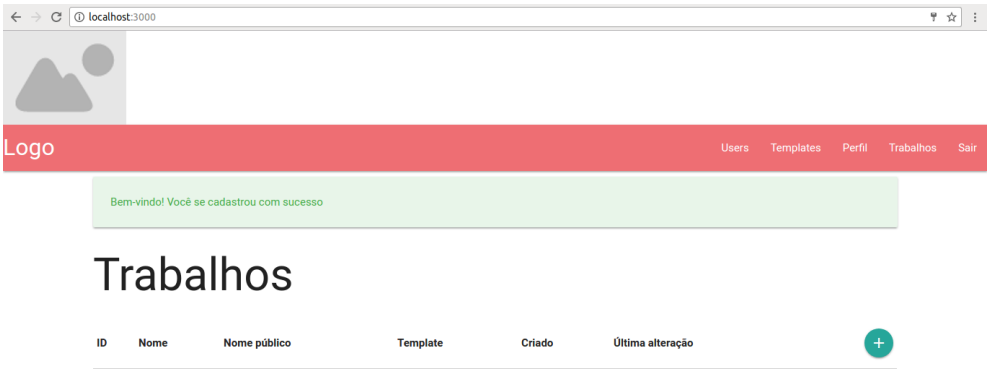
Figura 23 – Sistema: tela de cadastro




Fonte: os autores (2017)

Após o cadastro realizado, aparece um *toast* indicando o sucesso da operação, Figura 24, e então finalmente é mostrado a tela inicial do sistema para o usuário, que é justamente a listagem de documentos criados pelo usuário (um novo usuário obviamente irá aparecer uma lista em branco).

Figura 24 – Sistema: tela de trabalhos




Fonte: os autores (2017)

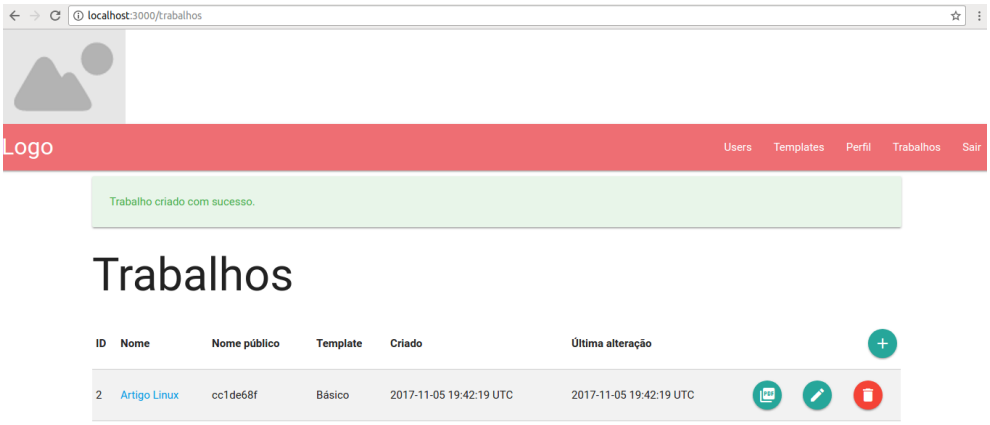
Para o usuário criar um novo documento, basta clicar no botão + (mais) para iniciar a criação de um novo documento. Na próxima tela, Figura 25, o usuário irá definir o nome do trabalho e também irá escolher o *template* que irá usar no documento. Após estas definições basta clicar no botão CRIAR TRABALHO.

Figura 25 – Sistema: tela de criação de trabalhos


Fonte: os autores (2017)

Automaticamente o usuário será redirecionado para a tela de listagem de trabalhos, que desta vez irá mostrar o trabalho criado, além de aparecer um *toast* de confirmação do novo trabalhado, Figura 26.

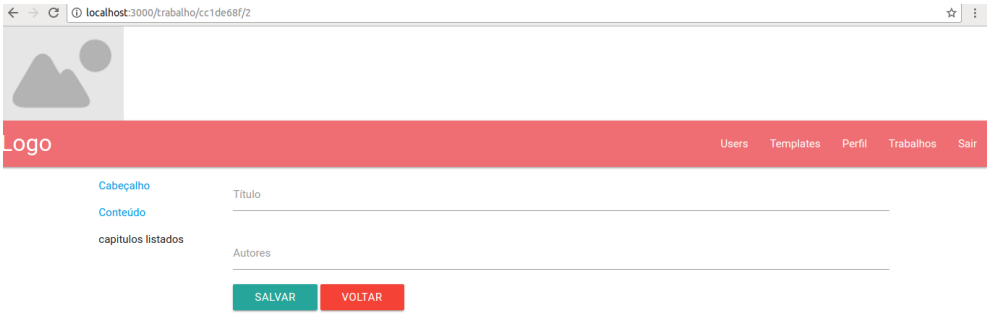
Figura 26 – Sistema: exibição do trabalho na tela de trabalhos




Fonte: os autores (2017)

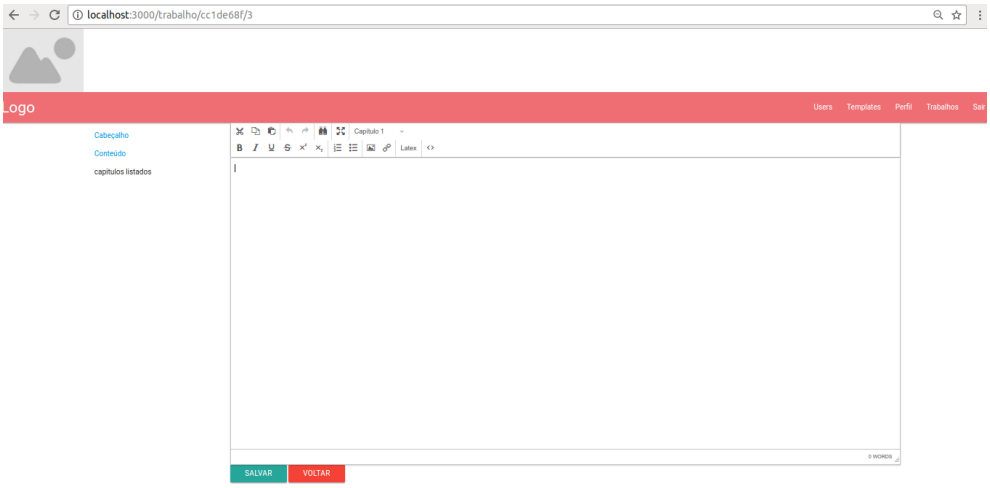
Ao clicar no nome do trabalho, o usuário irá para a tela onde é possível editar todas as áreas disponíveis do *template*. No *template* aqui mostrado existem duas áreas editáveis, uma que é o cabeçalho do documento, onde o usuário indica o título do documento e também o nome dos autores (2017), Figura 27, e outra área que é para editar o conteúdo do documento, Figura 28.

Figura 27 – Sistema: edição do trabalho utilizando os campos de edição




Fonte: os autores (2017)

Figura 28 – Sistema: edição de trabalho utilizando o editor de texto

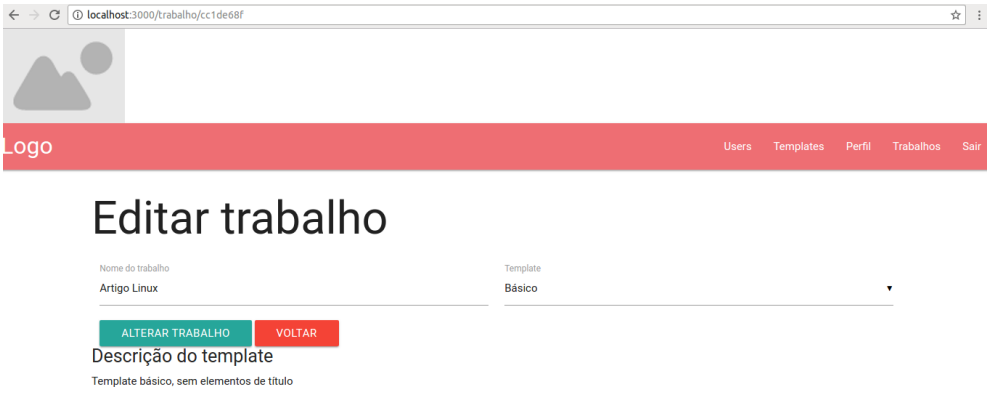



Fonte: os autores (2017)

O usuário após fazer todas as edições em todas as seções editáveis que tem acesso, pode salvar o conteúdo editado, este conteúdo será salvo no banco de dados para ser acessado posteriormente a qualquer momento pelo usuário.

Voltando a tela de listagem de trabalhos do usuário, uma ação que o usuário pode aplicar é a edição do tipo de trabalho clicando no ícone de edição do trabalho listado (ícone de lápis). Nesta edição é possível trocar o nome do trabalho, Figura 29, mas não o *template* utilizado para sua criação.

Figura 29 – Sistema: edição dos dados do trabalho

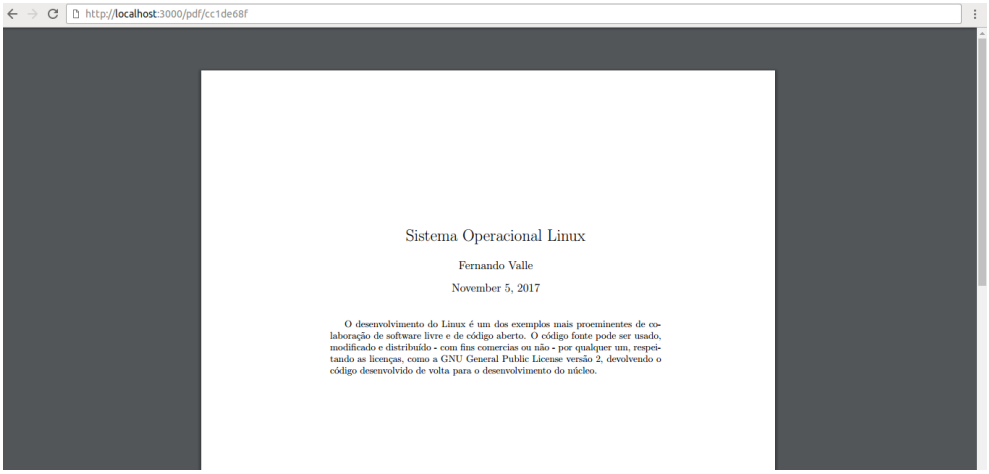



Fonte: os autores (2017)

Outra opção encontrada na listagem de trabalho é justamente a criação do PDF do documento, ou seja, a construção final do trabalho. Clicando no botão com

símbolo de PDF, o sistema irá compilar o documento e gerar o PDF final para o usuário, Figura 30.

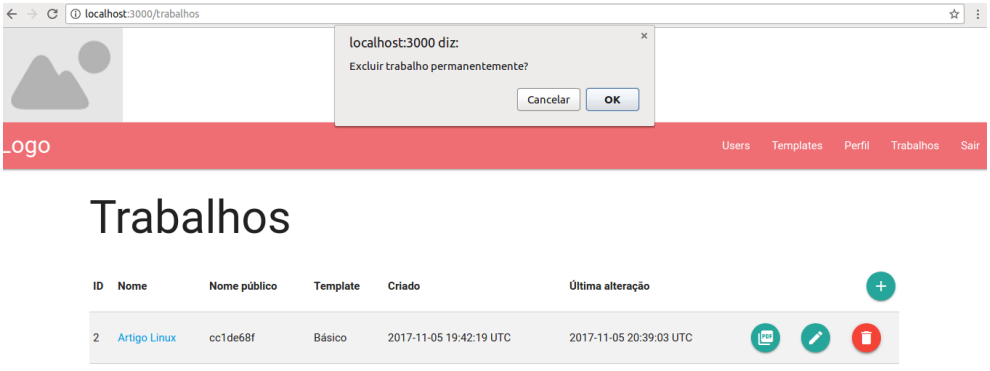
Figura 30 – Sistema: documento PDF gerado




Fonte: os autores (2017)

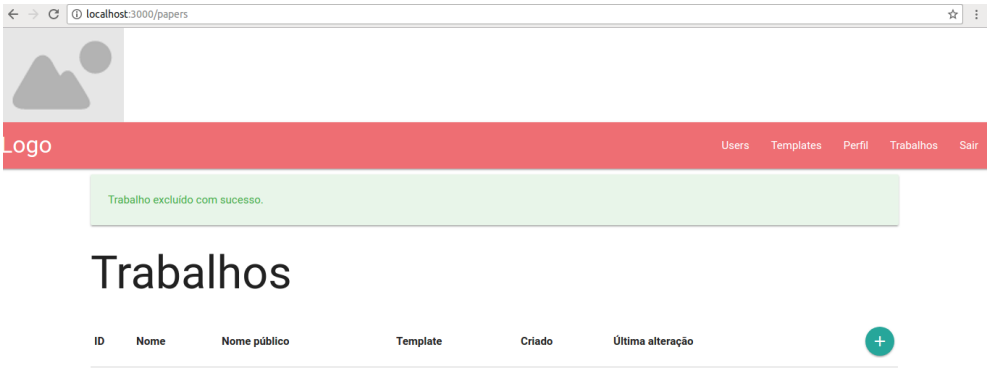
Outra opção que existe na listagem dos documentos é a exclusão do trabalho. O símbolo com ícone de lixeira deleta o documento do sistema. Após clicar na opção, surge uma mensagem para o usuário confirmar a exclusão do documento, Figura 31, caso confirme, um *toast* confirmando a opção aparece na tela de listagem de documentos, Figura 32.

Figura 31 – Sistema: confirmação da exclusão do trabalho




Fonte: os autores (2017)

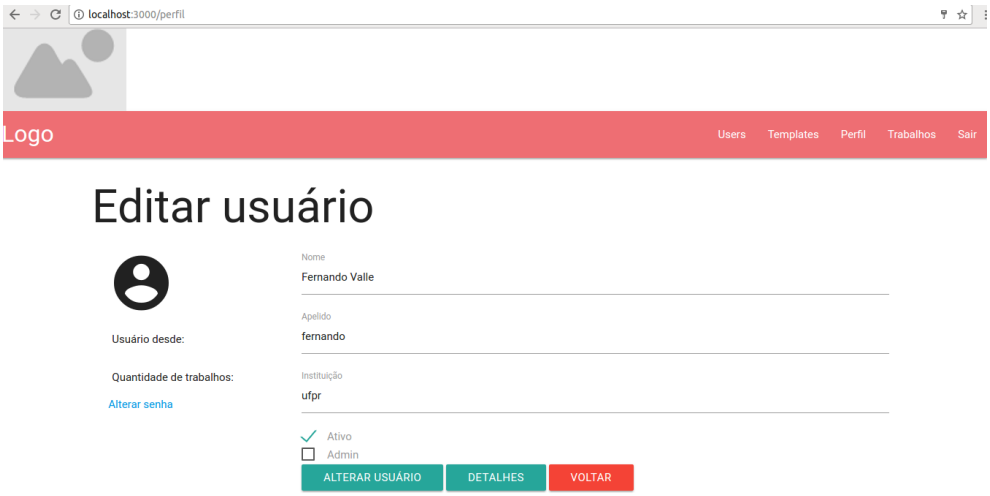
Figura 32 – Sistema: trabalho excluído




Fonte: os autores (2017)

A última opção que o usuário tem é a tela de edição de perfil, onde ele poderá alterar dados como seu nome, senha e instituição, como mostrado na Figura 33.

Figura 33 – Sistema: edição do perfil do usuário

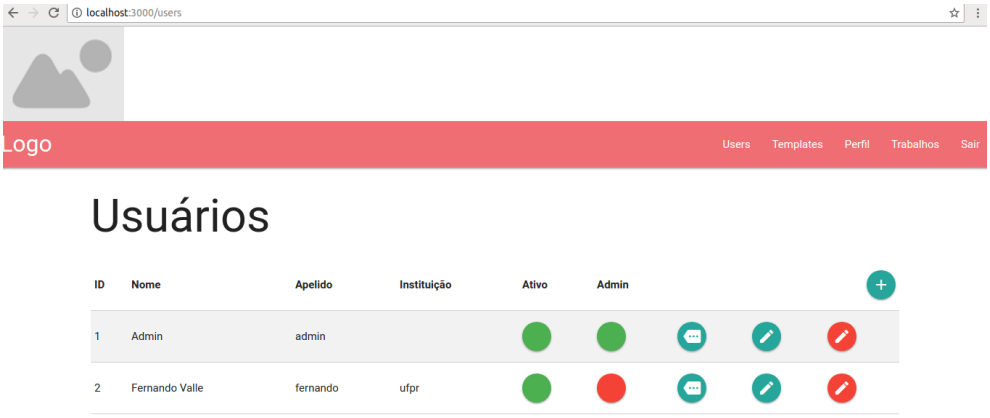



Fonte: os autores (2017)

### 4.2.2 ADMINISTRADOR

Tal qual o usuário, a rota padrão para iniciar a aplicação vai redirecionar o usuário para a listagem de seus trabalhos, logo após o *login*. Uma vez dentro de seu ambiente, o administrador terá algumas opções exclusivas liberadas para sua manipulação. Uma dessas ações é o item *Usuários*, ao clicar nesse item são listados todos os usuários cadastrados no sistema, Figura 34.

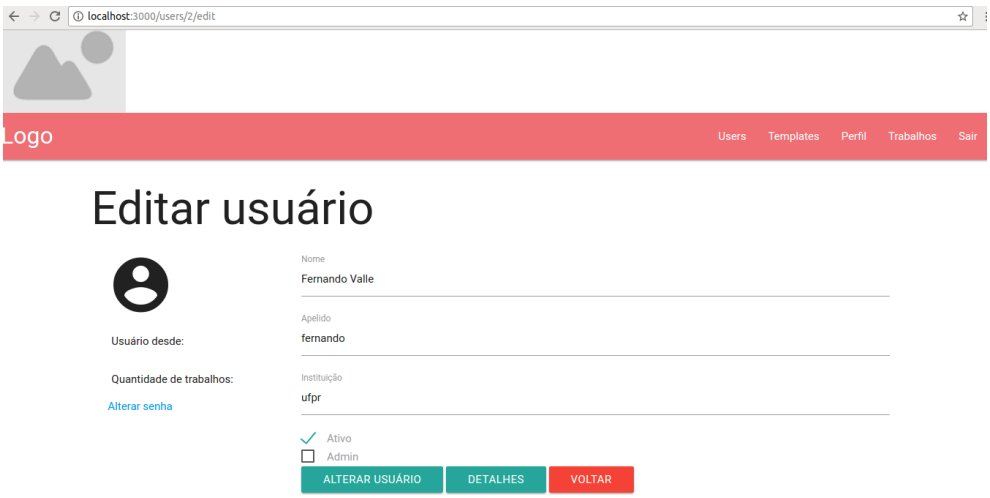
Figura 34 – Sistema: usuários cadastrados




Fonte: os autores (2017)

O administrador poderá escolher qualquer usuário listado e ao clicar no botão de edição (ícone de um lápis azul) poderá fazer alterações nesse usuário, Figura 35. Além de alterações básicas do perfil do usuário é possível torná-lo também um administrador, ativar ou desativar seu perfil e ver os detalhes de sua conta (a opção de detalhar a conta também está acessível na listagem de usuários cadastrados), Figura 36. Também é possível apagar um usuário ou mesmo criar um novo.

Figura 35 – Sistema: edição dos dados de usuários



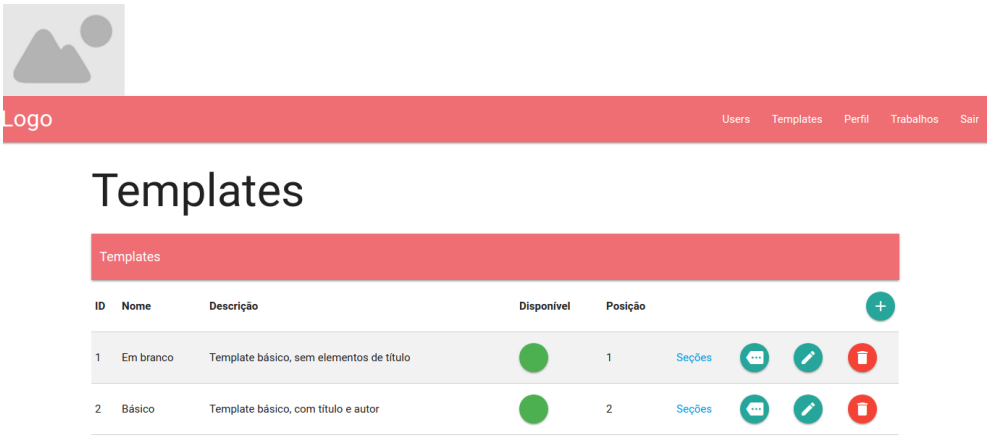

Fonte: os autores (2017)

Figura 36 – Sistema: detalhes do usuário


Fonte: os autores (2017)

Outra ação possível de ser feita pelo administrador é o gerenciamento dos *templates*. O *template* é o pontapé inicial para a criação de um trabalho. Para acessar a área de *templates* basta o administrador clicar no menu *Templates*. Será aberta a tela com a listagem de todos os *templates* criados no sistema, Figura 37.

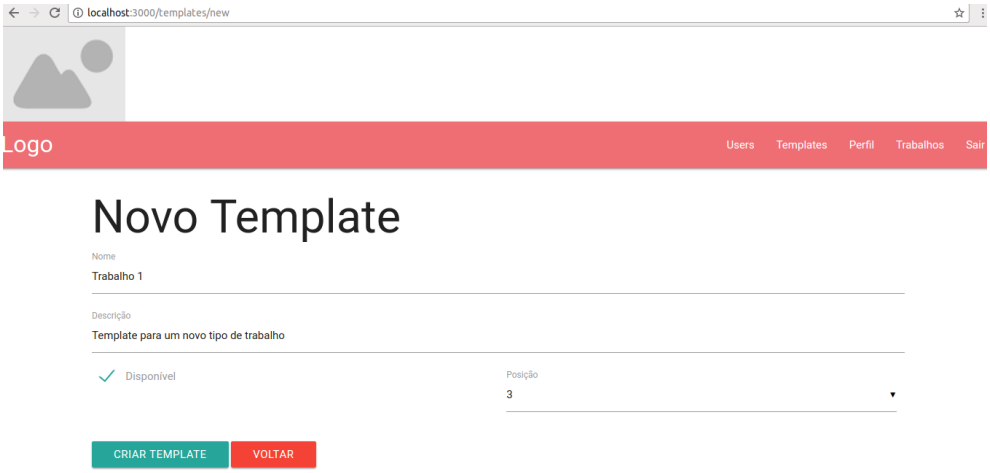
Figura 37 – Sistema: *templates* cadastrados




Fonte: os autores (2017)

Para criar um novo *template* basta clicar no botão novo (ícone de +) que será carregada a tela de criação, Figura 38. Inicialmente apenas o nome do *template*, uma descrição, a posição dele na listagem e uma confirmação de sua disponibilidade é exigida para sua criação.

Figura 38 – Sistema: novo *template*




Fonte: os autores (2017)

A partir desse momento é necessário um conhecimento básico de Latex para que seja utilizada toda a potencialidade da aplicação, pois agora serão criadas as seções do documento. Para isso basta clicar no item Seções na listagem de *templates*, a tela de edição de seções irá aparecer em branco, como mostrado na Figura 39.

Figura 39 – Sistema: Seções cadastradas


Fonte: os autores (2017)

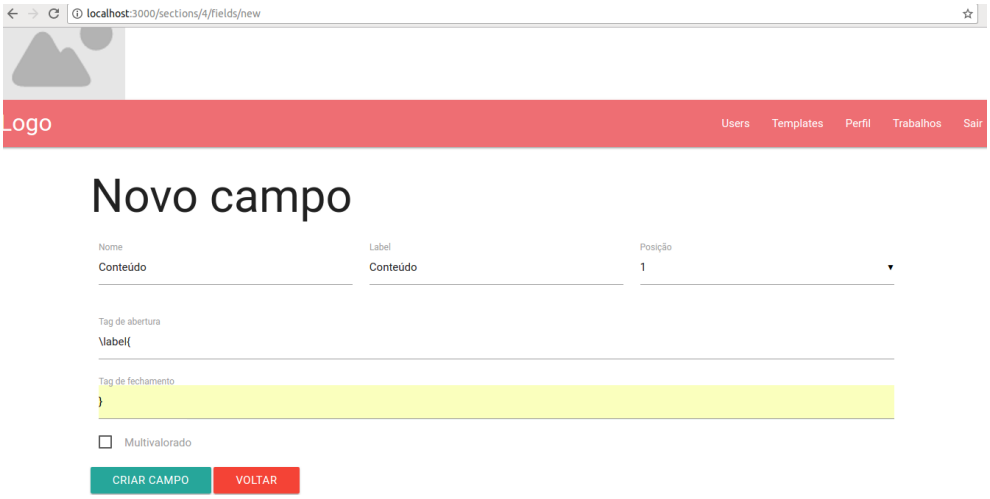
Na sequência, clica-se no botão de criar uma nova seção, a tela que será chamada é apenas para dar um nome a seção e escolher a posição que essa seção deverá aparecer na montagem do *template*, Figura 40. É muito importante a ordem da seção, pois, por exemplo, não é aconselhável, colocar a seção de conteúdo antes da parte pré-textual. Outro ponto que requer atenção aqui é a escolha entre o tipo de edição que a seção terá, se será editável ou não, e caso seja editável se terá campos a serem preenchidos por entradas de formulário ou será editável via editor de texto.

Figura 40 – Sistema: Nova seção


Fonte: os autores (2017)

Após a criação da seção, ela é listada na tela de seções, agora é possível clicar no item Campos, para fazer a edição dos campos da seção. Neste local deve-se colocar tags Latex para construir o *template*, Figura 41. Como uma seção pode ter vários campos, também deve-se ordenar em que momento deve aparecer o campo dentro da seção.

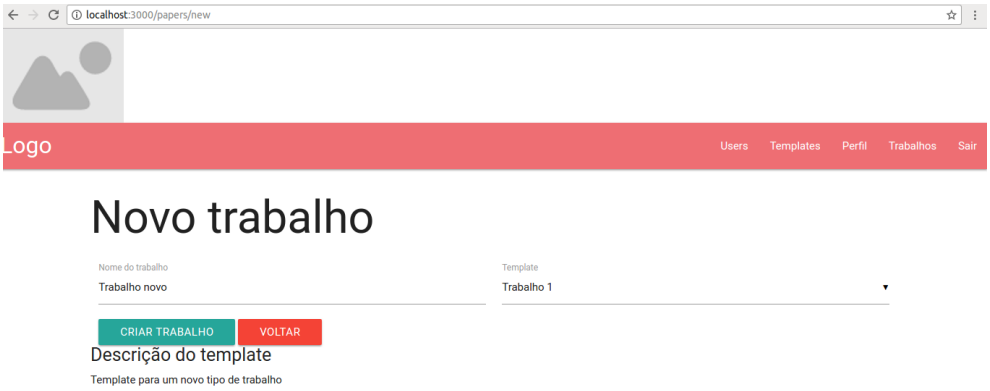
Figura 41 – Sistema: Novo campo




Fonte: os autores (2017)

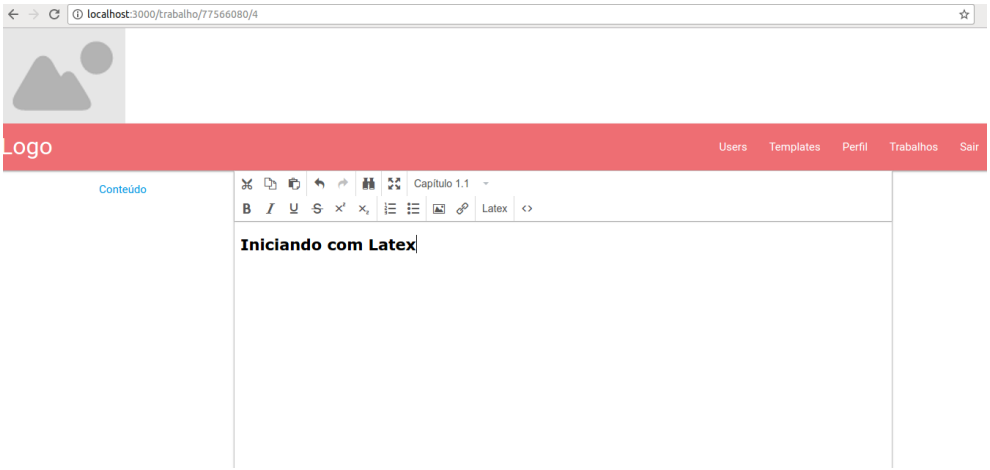
Com o *template* criado e configurado, já é possível qualquer usuário do sistema utilizá-lo para criar seu trabalho, Figuras 42 e 43.

Figura 42 – Sistema: utilizando *template* criado




Fonte: os autores (2017)

Figura 43 – Sistema: editando trabalho em novo *template*



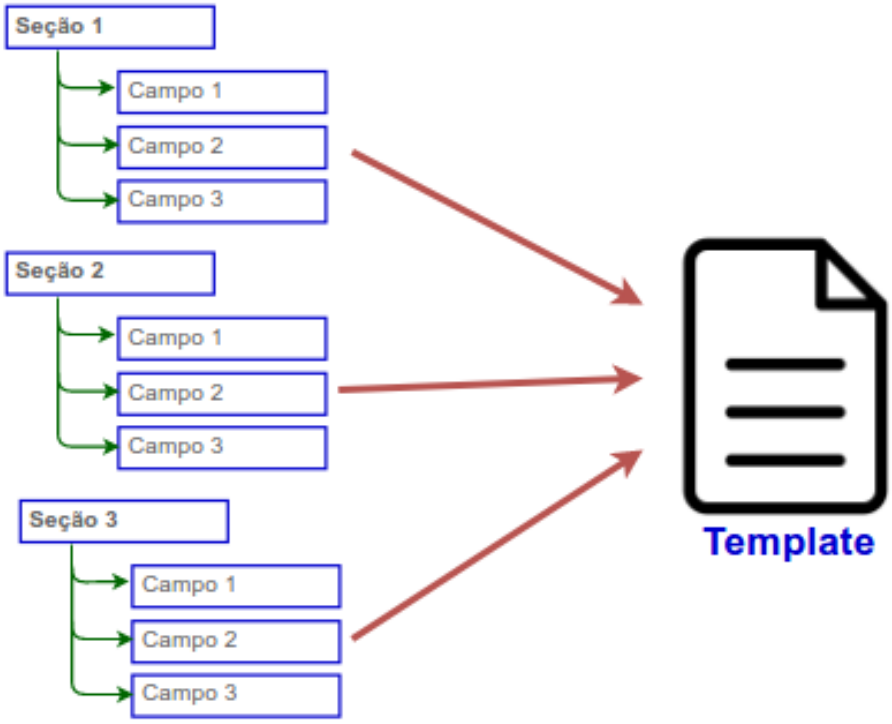

Fonte: os autores (2017)

### 4.2.3 CICLO DE VIDA DE UM DOCUMENTO

Um documento é um trabalho criado por um usuário, a geração desse documento demanda várias etapas até sua conclusão.

1 – Criação do *template*: o *template* selecionado pelo usuário é previamente criado por um administrador. Esse *template* é criado seguindo as regras de criação de documentos Latex. Na etapa de criação do *template* o administrador irá inserir códigos Latex puros para a adequação a proposta de *template* criado. Nesta etapa são geradas seções do documento e dentro dessas seções seus campos para serem preenchidos (ou não) pelo usuário, a Figura 44 ilustra isso.

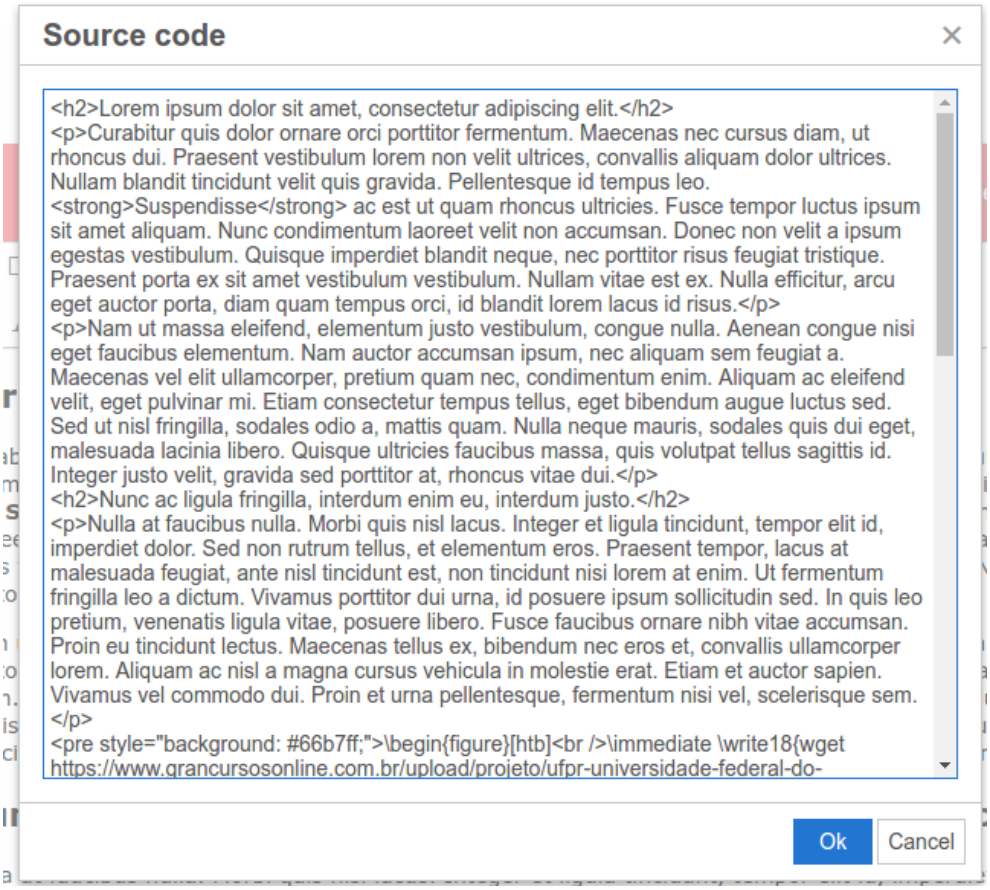
Figura 44 – Estrutura de um documento




Fonte: os autores (2017)

2 – Conteúdo do documento: após definido pelo usuário o *template* desejado, será inserido o conteúdo do trabalho. Existem áreas de conteúdo que seu preenchimento será feita via campos de formulário. Porém existem também áreas onde será carregado um editor de textos para o usuário escrever. O editor é um *fork* do *tinymce*, popular editor HTML de textos, com isso, toda a formatação que o usuário faça no documento usando o editor será feita em código HTML. Com isso temos o conteúdo do trabalho do usuário todo formatado em HTML, Figura 45.

Figura 45 – Marcação HTML no editor de texto




Fonte: os autores (2017)

3 – Tradução do documento: quando o usuário executa a ação para gerar o documento final no formato PDF, é feita a tradução do conteúdo HTML para Latex, que é a linguagem que será compilado o trabalho. Quem faz essa tarefa de tradução é uma *Gem* chamada Html2latex, Figura 46.

Figura 46 – Conversão HTML para Latex


Fonte: os autores (2017)

A *Gem* tem uma determinada lista de *tags* HTML fazendo correspondência a comandos Latex, então, por exemplo quando um determinado trecho de texto é encontrado pela *Gem* com a *tag* <b> (que é a formatação em HTML para negrito), a *Gem* substitui essa *tag* pelo comando *textbf*. Através do console do *Rails* é possível utilizar a *Gem* para verificar sua usabilidade via chamadas de terminal, podemos por exemplo fazer uma tradução de um trecho qualquer escrito em HTML:

```
2.3.3 :001 > a = "<b>Negrito!</b>"
 => "<b>Negrito!</b>"
2.3.3 :002 > Html2latex.translate(a)
 => "\\textbf{Negrito!}"
```

Existem ainda outras funcionalidades na *Gem* como inserir novas *tags* para tradução e exclusão de *tags*, todas as suas funcionalidades encontram-se na documentação oficial[1].

4 – Compilação para geração do PDF final: a etapa final para geração do documento é sua compilação, que é feita com o software livre *pdflatex*, que encontra-se dentro do programa *texlive* (versão *full*). Quem chama a compilação é o *controller* build_controller.rb. Nesse *controller* existem uma função chamada *system* chamada da seguinte forma:

```
system("pdflatex --interaction=nonstopmode -output-directory=public/pdf
--shell-escape #{arquivo}")
```

Explica-se o comando:

[1] *https://rubygems.org/gems/html2latex/*

- pdflatex – o programa que compila o arquivo .Tex gerado pelo tradutor em .pdf
- –iteraciton=nonstopmode – por padrão todos os *warning* que o *texlive* faz para a execução pede uma ação do usuário, com esse parâmetro, os *warnings* são resolvidos sem a necessidade de intervenção do usuário.
- -output-directory=publicpdf – local onde será armazenado o arquivo gerado
- –shell-escape – irá escapar eventuais comandos de terminal que estejam dentro do documento para que sejam executados.
- #{arquivo} – variável do *Ruby* com o nome do arquivo a ser gerado.

# 5 CONSIDERAÇÕES FINAIS

Esse trabalho procurou dar conta de uma série de objetivos que, analisados conjuntamente, tiveram sua conclusão bem sucedida. O software desenvolvido - AbnTexTor - se mostra funcional, versátil e ao mesmo tempo complexo o suficiente para que seu desenvolvimento posterior possa se tornar uma aplicação de mercado. Na utilização do software, o usuário edita apenas o conteúdo referente a cada parte do trabalho, deixando a formatação inteiramente para o sistema, cumprindo o objetivo geral proposto.

Dentro do conjunto dos objetivos específicos, os itens elencados também foram cumpridos. Destaca-se: a análise das ferramentas de mercado feita no Capítulo 2, analisando mais de 12 ferramentas em diferentes nichos - edição tradicional de documentos, edição em nuvem, edição em Latex - e mostrando o posicionamento do AbnTeXtor frente a esse mercado, que invariavelmente deixa a cargo do usuário a responsabilidade da formatação dos trabalhos; a construção do editor de texto proposto também foi possível utilizando ferramentas de edição *WYSIWYG*, que gera a marcação de texto em linguagem HTML que é posteriormente convertida para Latex com a *Gem* desenvolvida pelo grupo; a infraestrutura desenvolvida para acomodar a proposta do editor de texto e as regras de formatação dentro do próprio sistema, apesar de mínima, mostra-se flexível, porém complexa - a estrutura do sistema é abordada no capítulo 4 e seu desenvolvimento no Capítulo 3.

A construção do algoritmo de conversão HTML para Latex é um dos principais pontos positivos do sistema desenvolvido. Isso porque, dada a grande e conhecida profusão de ferramenta de terceiros utilizadas na linguagem Ruby, as chamadas *gems*, não foi possível localizar uma que fizesse essa tarefa, nem no repositório oficial, nem em repositórios do *github*. Além disso, a gema foi publicamente hospedada no dia 8 de novembro e em 24 horas possuía mais de 150 downloads.

Por fim, a documentação do sistema está exposta nesse trabalho em forma de apêndices e são comentadas ao longo do Capítulo 3, conforme o sistema foi sendo desenvolvido e a documentação gerada.

Alguns elementos planejados para as funcionalidades do editor deixaram de ser implementadas, conforme o prazo se esgotava. Isso pode ser observado pela diferença entre o Apêndice A (DER) e o Apêndice G (DER AJUSTADO). Duas das funcionalidades não desenvolvidas foram a inclusão de um subsistema de gerenciamento de imagens para uso nos trabalhos e um subsistema de gerenciamento e inclusão de referências. As duas funcionalidades planejadas no escopo inicial do trabalho mostraram demandar

um tempo de desenvolvimento muito grande, além de terem que ser definidas questões técnicas como armazenamento, processamento, acesso, etc. Considerou-se, entretanto, que a ausência dessas funcionalidades não afetariam a proposta de um editor de texto para conversão HTML para Latex, como pode ser observado pelo funcionamento do sistema.

A vontade de implementar nova ideias ao longo do desenvolvimento do trabalho foi grande, mas incorria-se sempre no risco de alargar o escopo e sem prejuízo aos prazos. Diante desse cenário, buscou-se ao máximo cumprir o que foi estabelecido o inicio do projeto e deixando novas ideias para um momento posterior. Algumas questões debatidas sobre esse tema foram:

- Alterações na estrutura da modelagem de dados, fazendo relacionamento muitos-para-muitos entre *Template* e *Section* e entre *Section* e *Field*, para que uma *Section* pudesse ser usada por mais de um *Template* e uma *Field* pudesse ser usado por mais de uma *Section*. Essa alteração é particularmente importante para que o usuário possa alterar o tipo de trabalho (*Template*) sem perder o conteúdo já escrito no *Template* inicial.
- Compartilhamento de trabalhos e edição simultânea, como ocorre em alguns editores de texto em nuvem.
- Aumentar a versatilidade do editor de texto, tendo a possibilidade de utilizar opções customizadas para cada *Template* ou *Section*, sendo que essa versatilidade também tem que estar refletida na *gem*.
- Desenvolver um subsistema de referências bibliográficas para serem usadas durante a edição do trabalho, tornando o produto final mais completo.
- Desenvolver um subsistema de gestão de imagens para serem utilizadas durante a edição do trabalho, minimizando a necessidade de escrita de código Latex para o usuário final.
- Experiência do usuário, com o desenvolvimento de funcionalidades que sejam executadas no cliente da aplicação, como salvamento automático e periódico do trabalho, carregamento de segundo plano por meio de requisições assíncronas - uma vez que o desenvolvimento teve seu foco nas funcionalidades executadas pelo servidor da aplicação.
- Testar outras opções de editores de texto HTML para serem utilizadas na página de edição de conteúdos, procurando maior flexibilidade e estabilidade para a ferramenta.

- Paralelizar o processo de compilação e apresentação do trabalho ao usuário para que as chamadas as sistema não inviabilizem o uso do aplicativo durante esse processo.

Essas e muitas outras funcionalidades podem ser desenvolvidas em um futuro próximo, por qualquer pessoa interessada no projeto. A *Gem* está disponível online[1] e o código fonte do sistema está disponível juntamente com esse trabalho.

[1] *https://rubygems.org/gems/html2latex/*

# REFERÊNCIAS

BECK, K. et al. **O Manifest Ágil**. [S.l.], 2014. Disponível em: <http://www.manifestoagil.com.br/>. Citado na página 36.

BOOCH, G. **UML: guia do usuário**. Rio de Janeiro: Elsevier, 2012. ISBN 978-85-352-1784-1. Citado 2 vezes nas páginas 46 e 47.

COHN, M. **Desenvolvimento de Software com Scrum: aplicando métodos ágeis com sucesso**. Porto Alegre: Bookman, 2011. ISBN 978-85-7780-807-6. Citado 2 vezes nas páginas 37 e 38.

DEBONI, J. E. Z. **Modelagem orientada a objetos com UML**. São Paulo: Futura, 2003. ISBN 85-7413-166-0. Citado na página 47.

FUENTES, V. B. **Ruby on Rails: Coloque sua aplicação web nos trilhos**. São Paulo: Casa do Código, 2012. Citado na página 50.

KNUTH, D. E. **The TeXbook**. Massachusetts: Addison-Wesley, 1986. ISBN 0-201-13448-9. Disponível em: <http://www.ctex.org/documents/shredder/src/texbook.pdf>. Citado na página 19.

STALLMAN, R. **Why Open Source misses the point of Free Software**. [S.l.], sem data. Disponível em: <https://www.gnu.org/philosophy/open-source-misses-the-point.html>. Citado na página 20.

VIREIRA, D. **Scrum: a metodologia ágil excplicada de forma definitiva**. [S.l.], 2014. Disponível em: <http://www.mindmaster.com.br/scrum/>. Citado 2 vezes nas páginas 36 e 37.

# APÊNDICES

# APÊNDICE A – DER

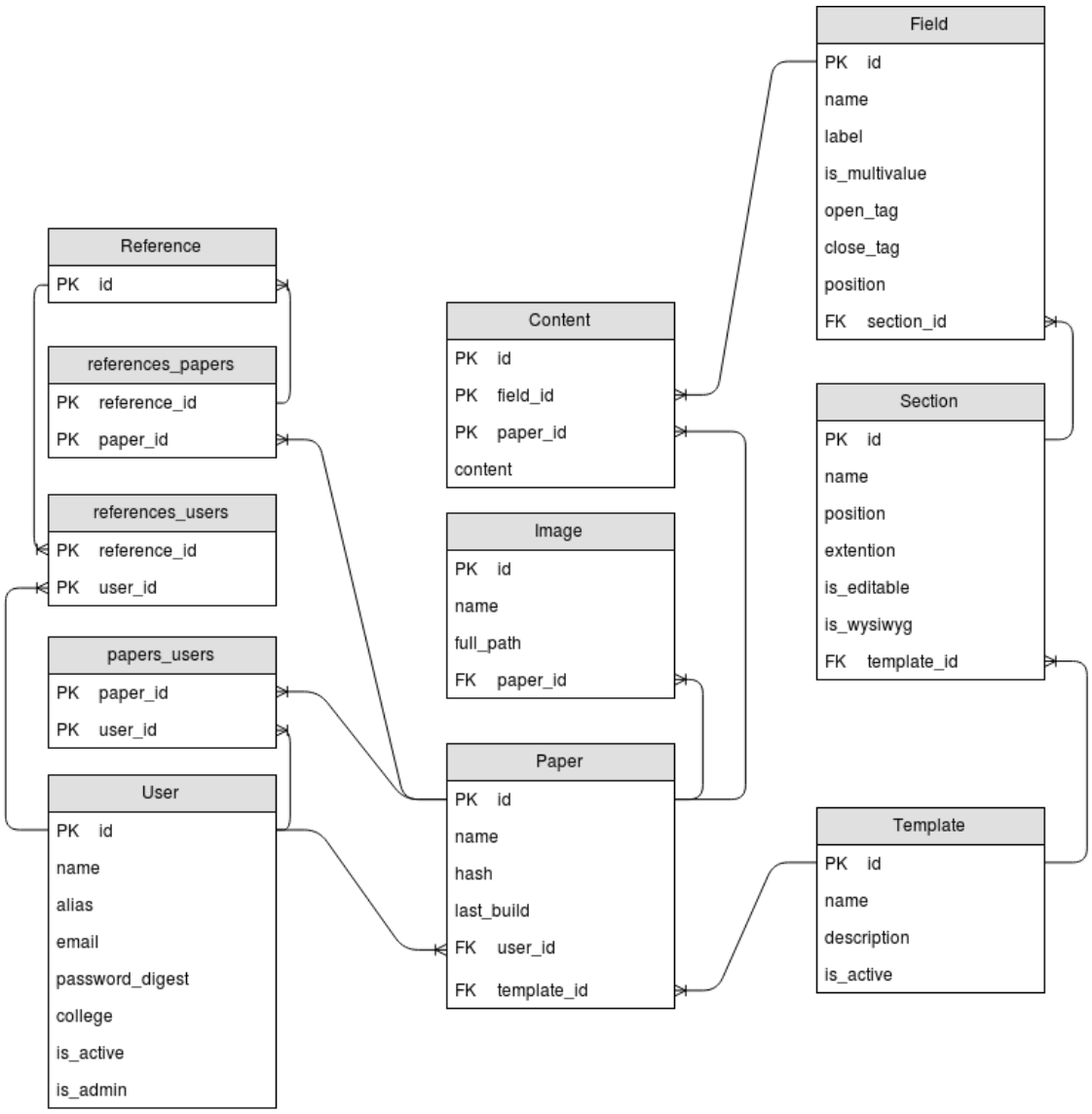
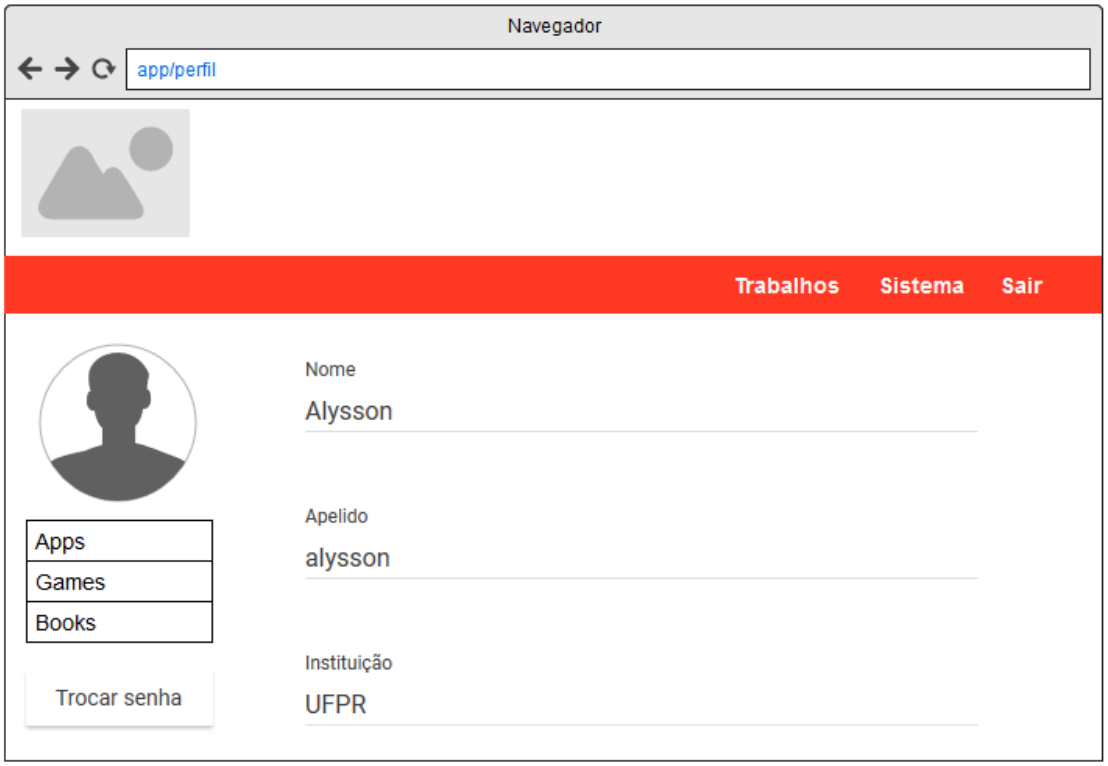



Figura 47 – Diagrama Entidade Relacionamento

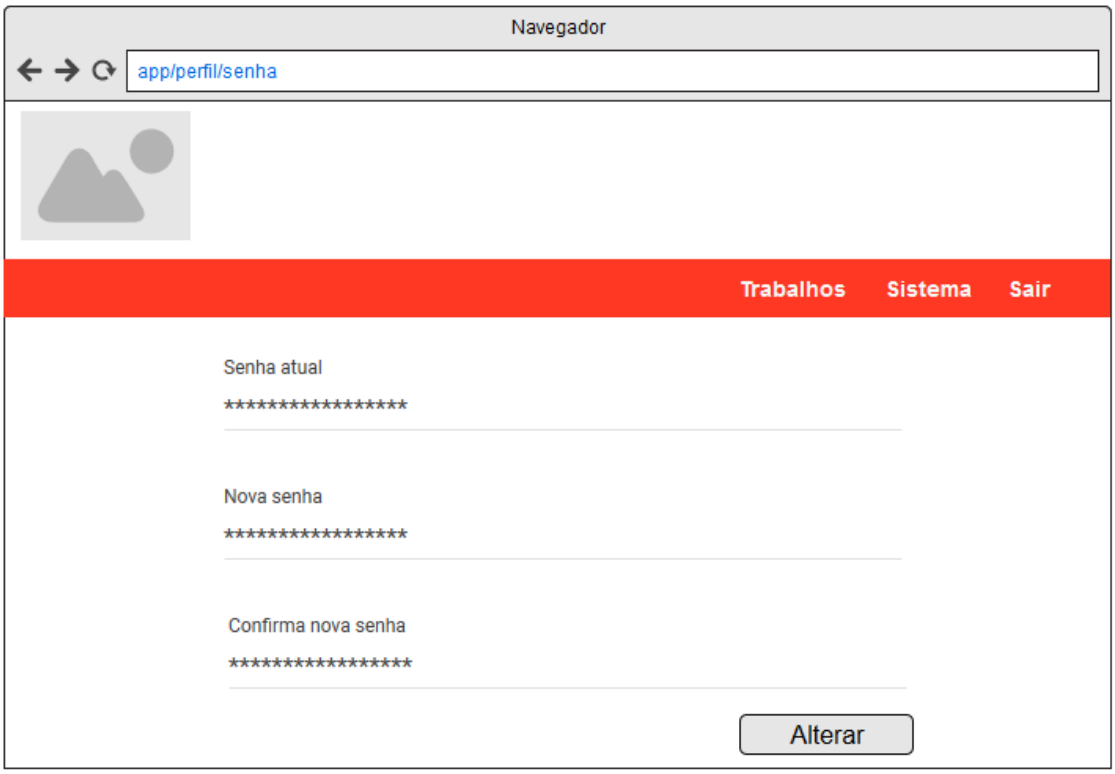
# APÊNDICE B – PROTÓTIPOS DAS TELAS

Figura 48 – Tela do usuário: perfil



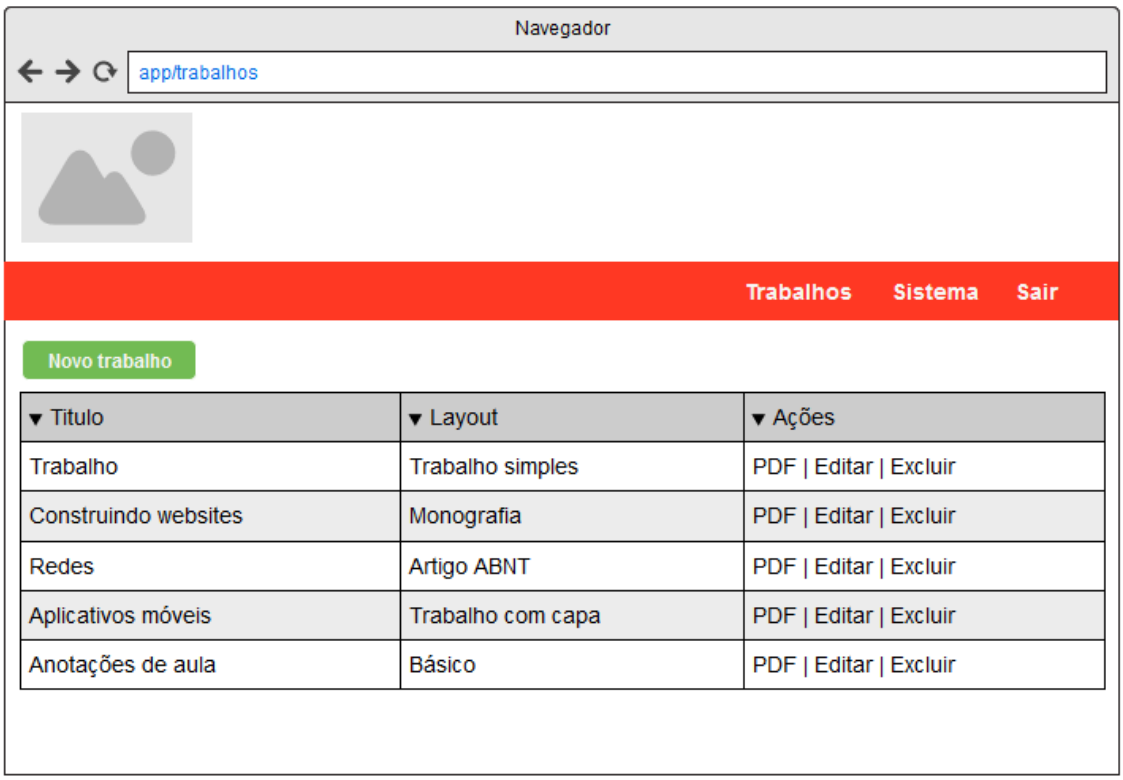

Fonte: os autores (2017)

Figura 49 – Tela do usuário: alteração de senha



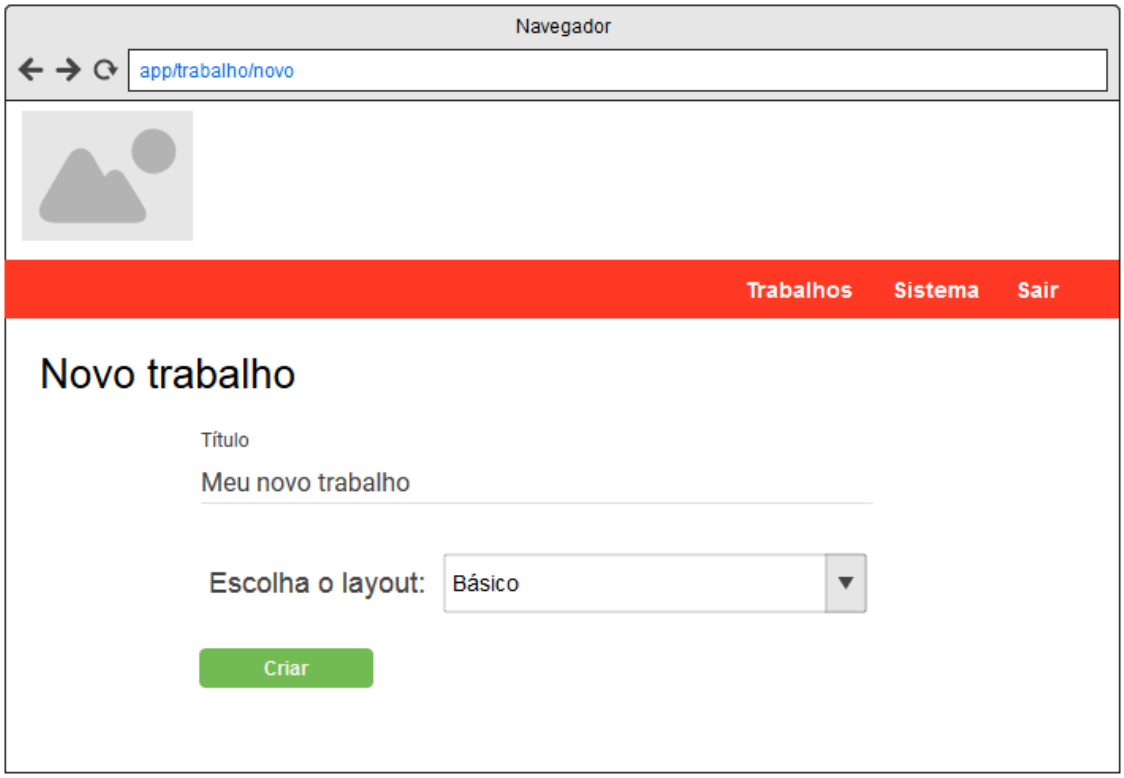

Fonte: os autores (2017)

Figura 50 – Tela do usuário: trabalhos



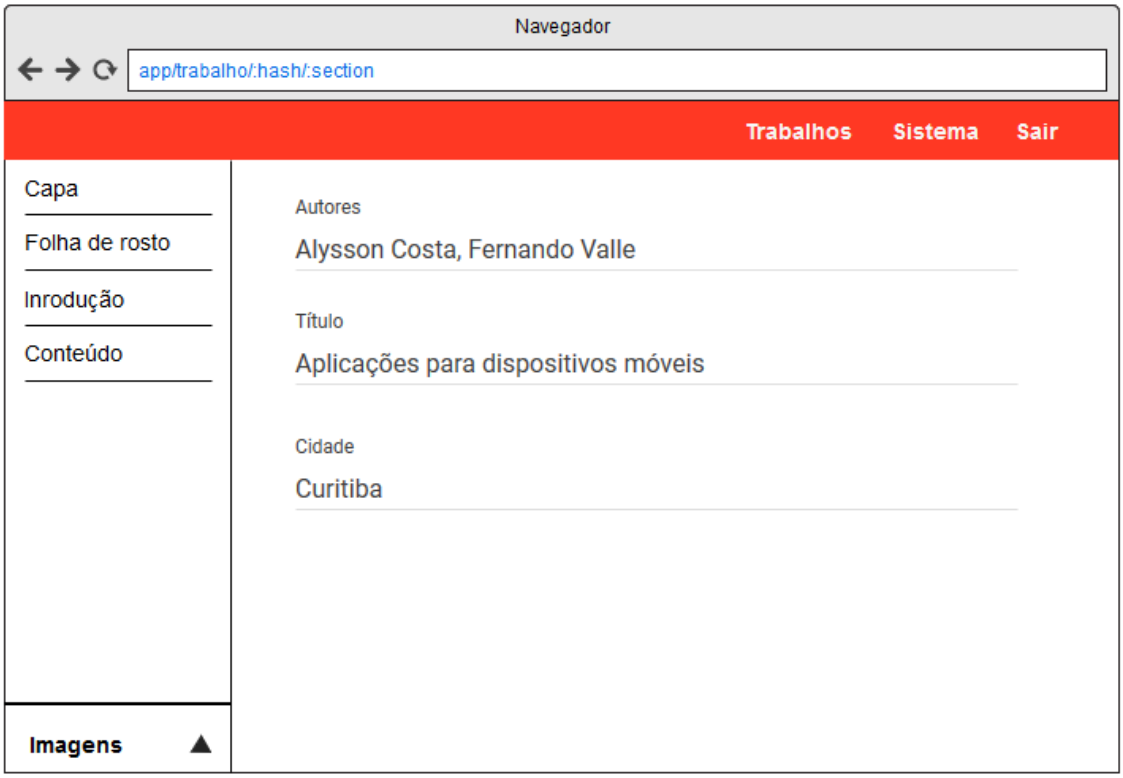

Fonte: os autores (2017)

Figura 51 – Tela do usuário: novo trabalho



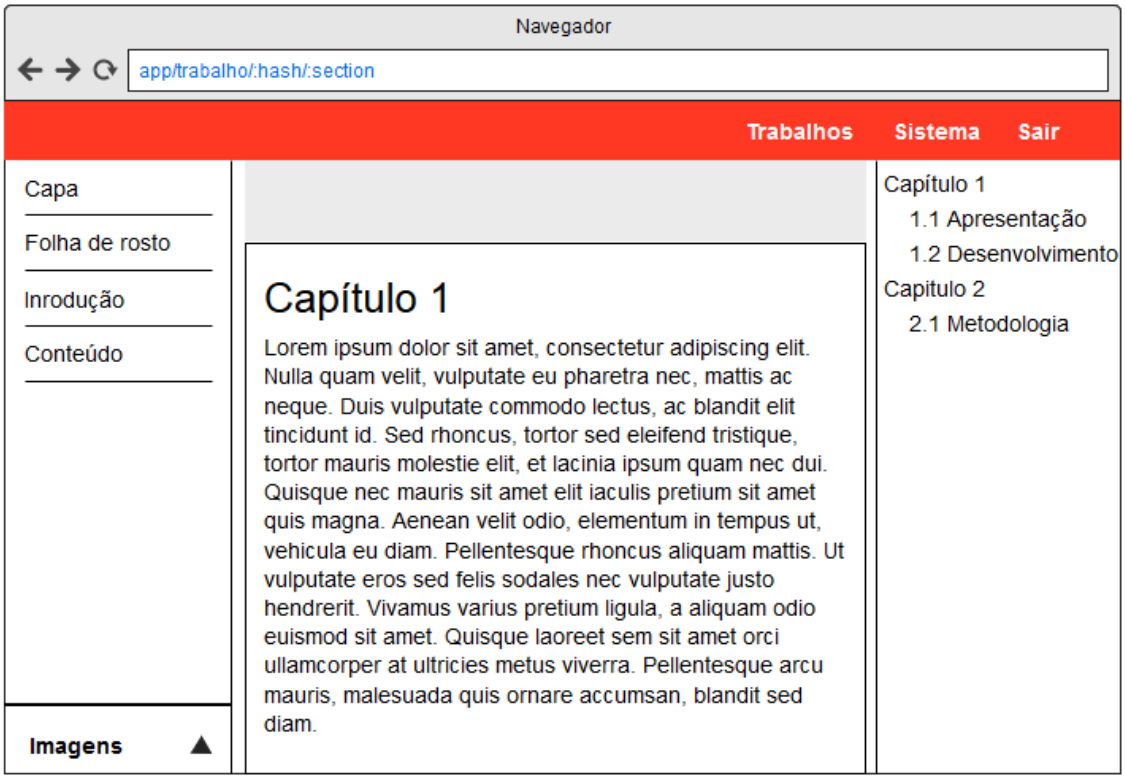

Fonte: os autores (2017)

Figura 52 – Tela do usuário: edição de campos do trabalho



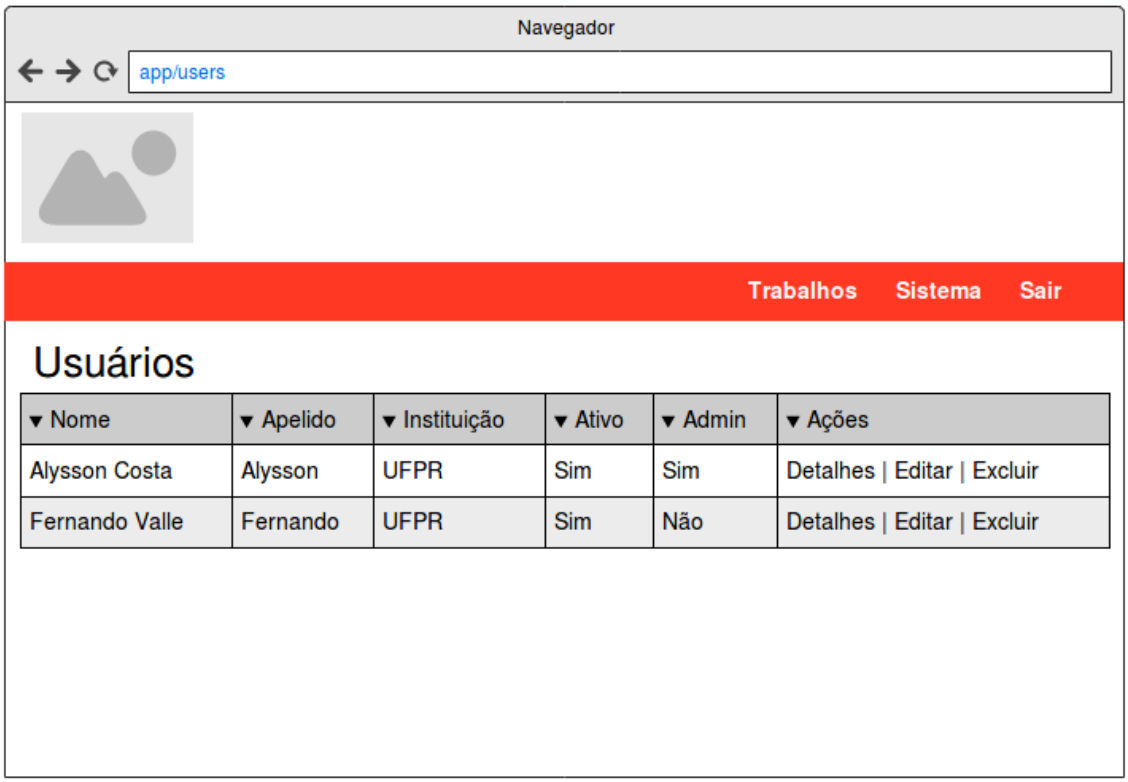

Fonte: os autores (2017)

Figura 53 – Tela do usuário: editor de texto do trabalho



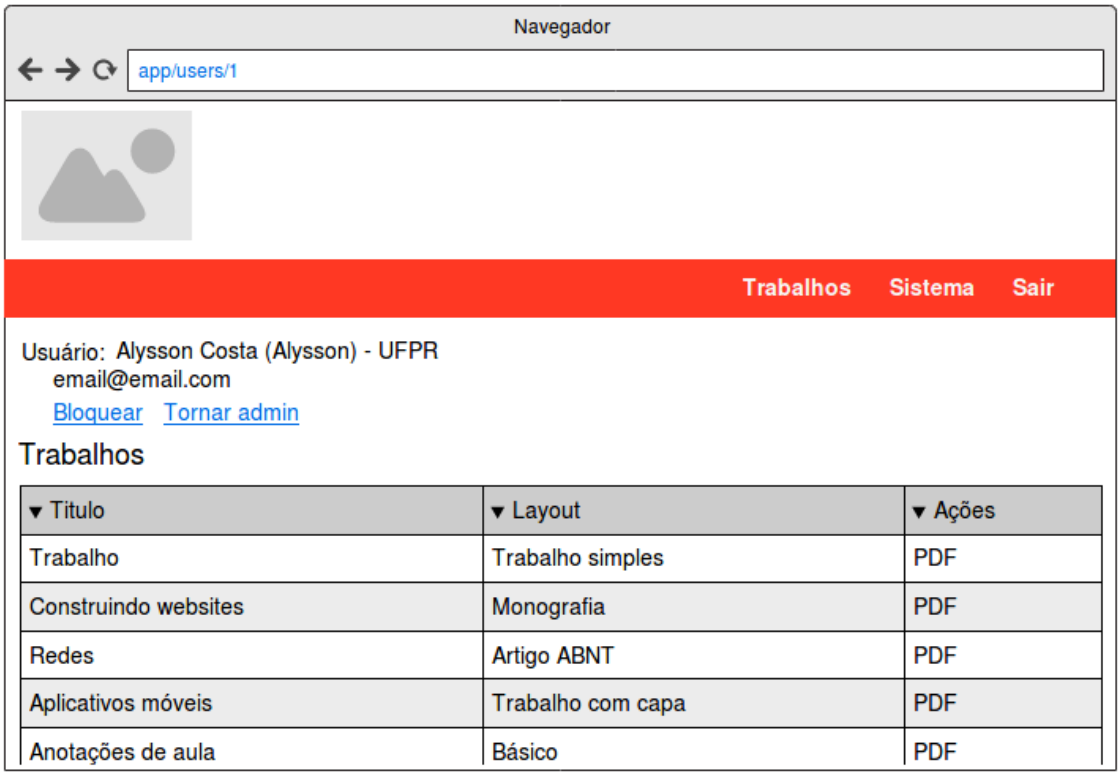

Fonte: os autores (2017)

Figura 54 – Tela do sistema: usuários




Fonte: os autores (2017)

Figura 55 – Tela do sistema: detalhes do usuário

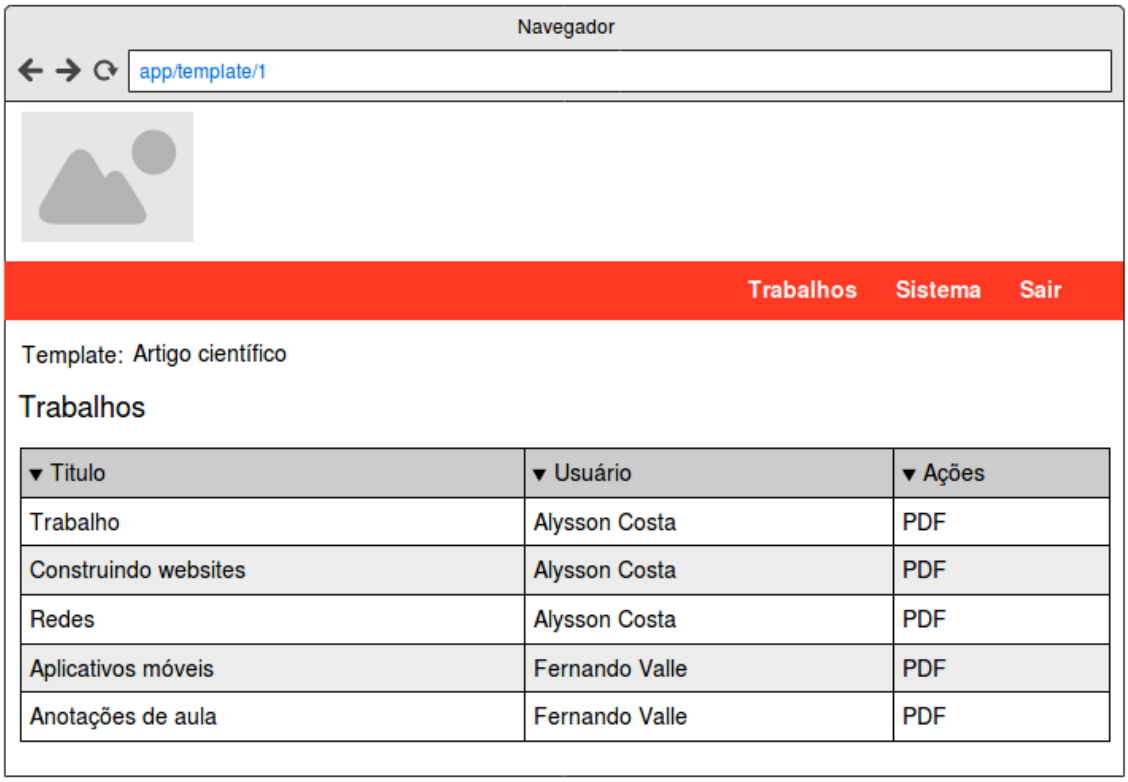



Fonte: os autores (2017)

Figura 56 – Tela do sistema: templates

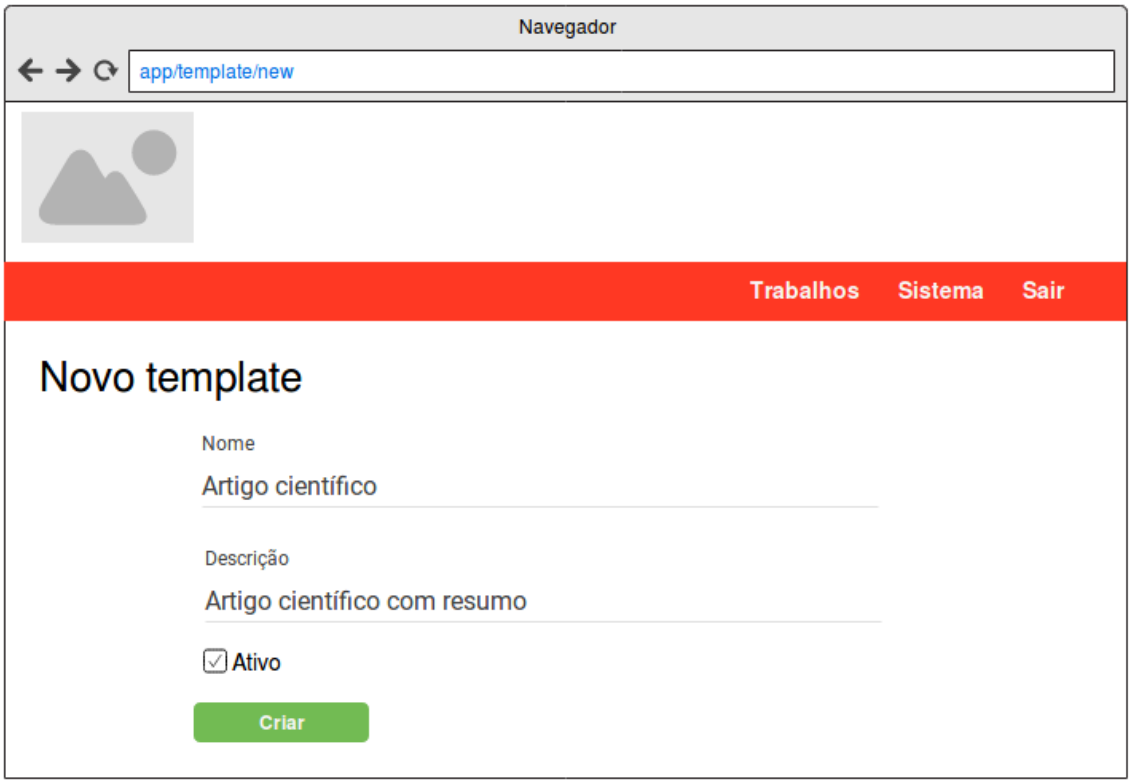

Fonte: os autores (2017)

Figura 57 – Tela do sistema: detalhes do template



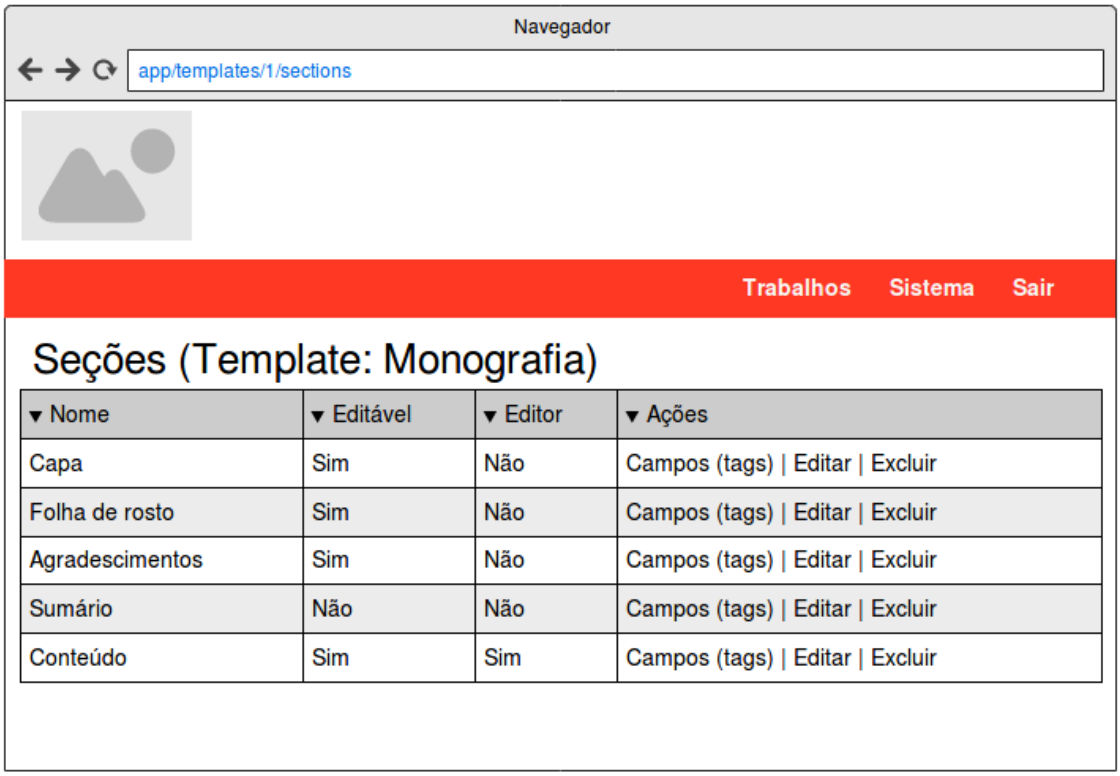

Fonte: os autores (2017)

Figura 58 – Tela do sistema: criação e edição de template



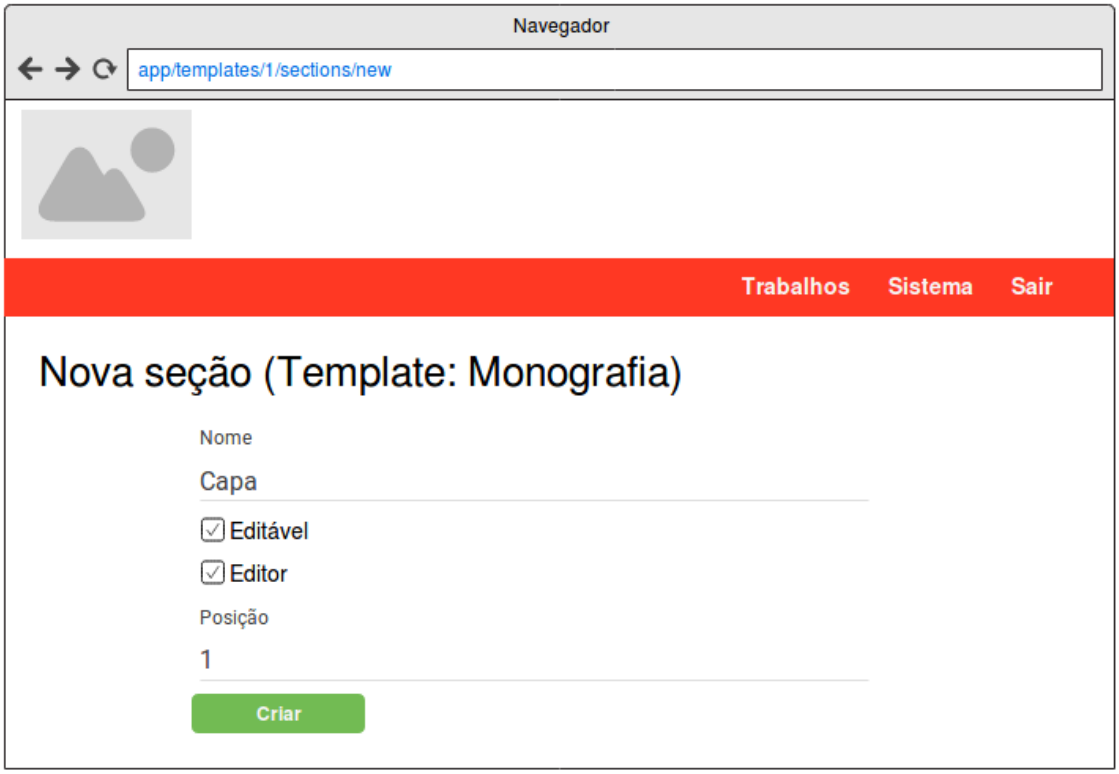

Fonte: os autores (2017)

Figura 59 – Tela do sistema: seções



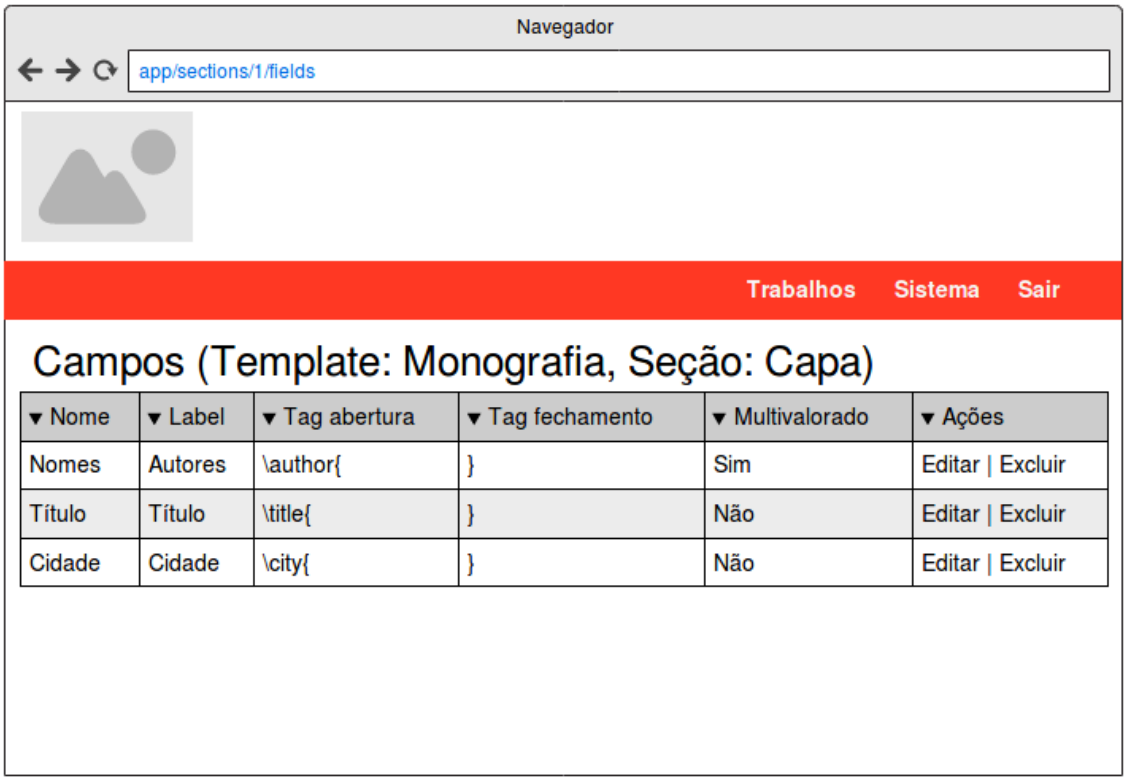

Fonte: os autores (2017)

Figura 60 – Tela do sistema: criação e edição de seção



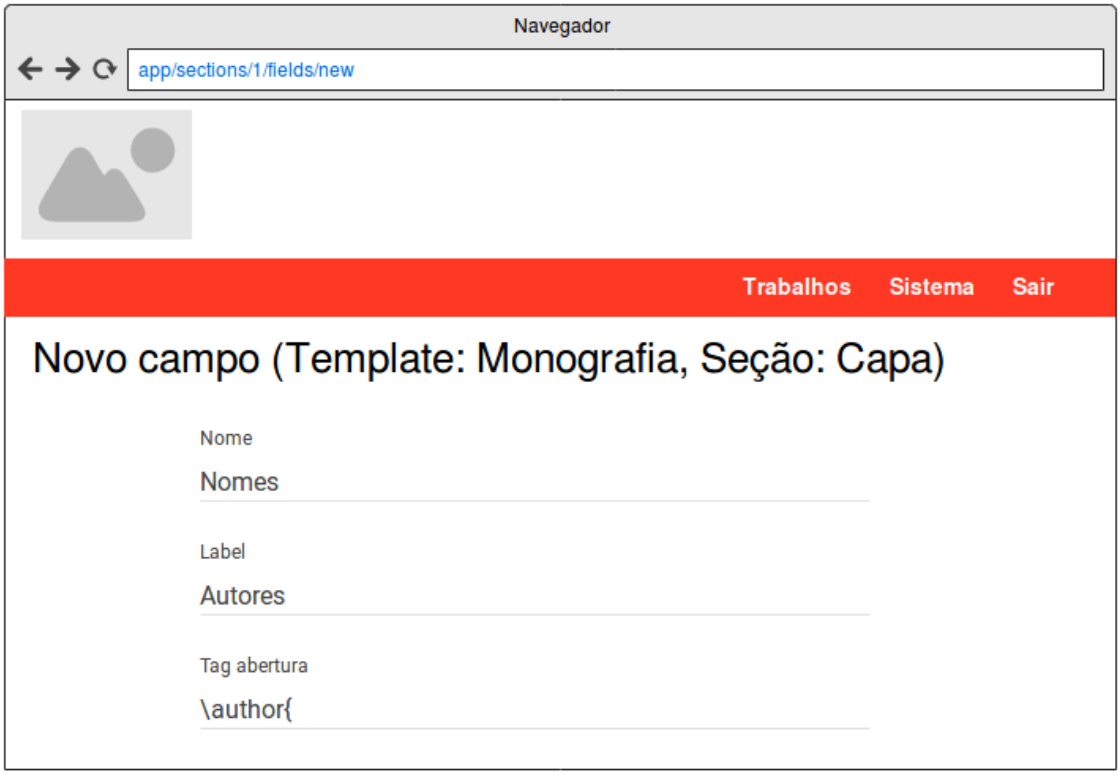

Fonte: os autores (2017)

Figura 61 – Tela do sistema: campos




Fonte: os autores (2017)

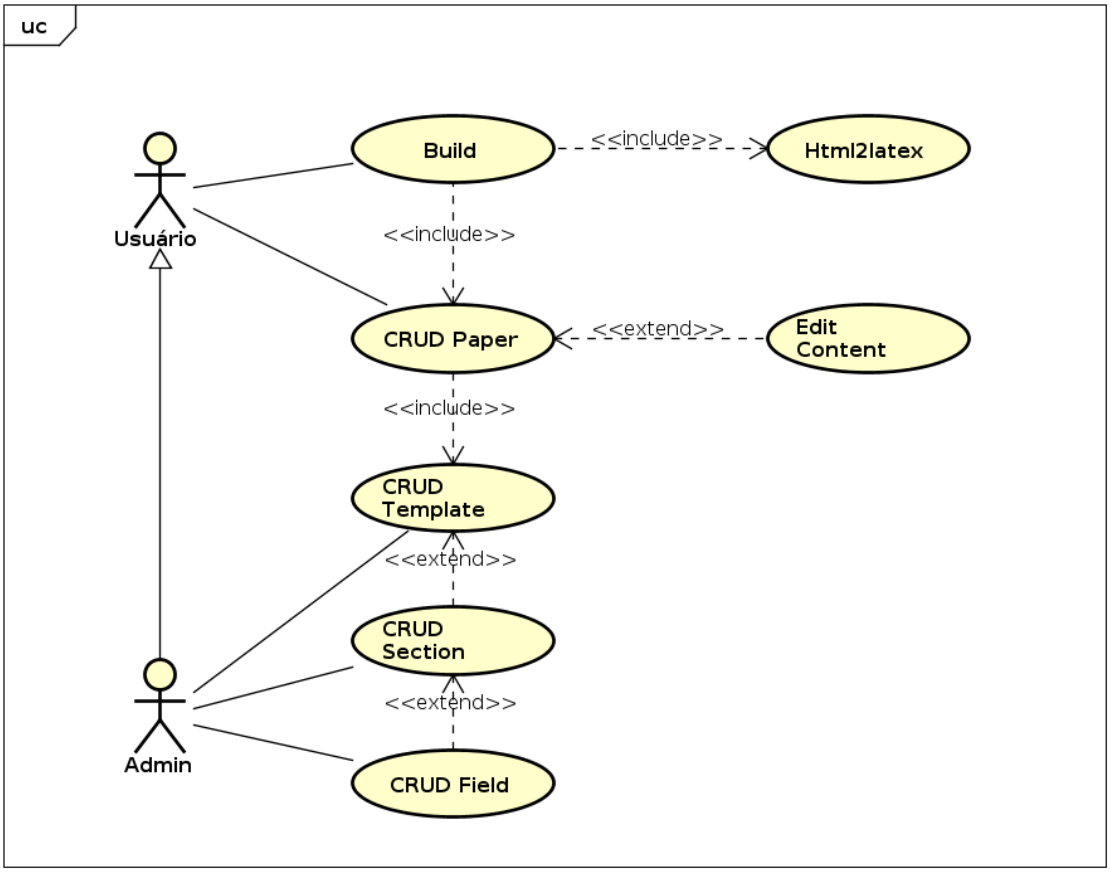
Figura 62 – Tela do sistema: criação e edição de campos




Fonte: os autores (2017)

# APÊNDICE C – DIAGRAMA DE CASOS DE USO

Figura 63 – Diagrama de Casos de Uso - Simplificado

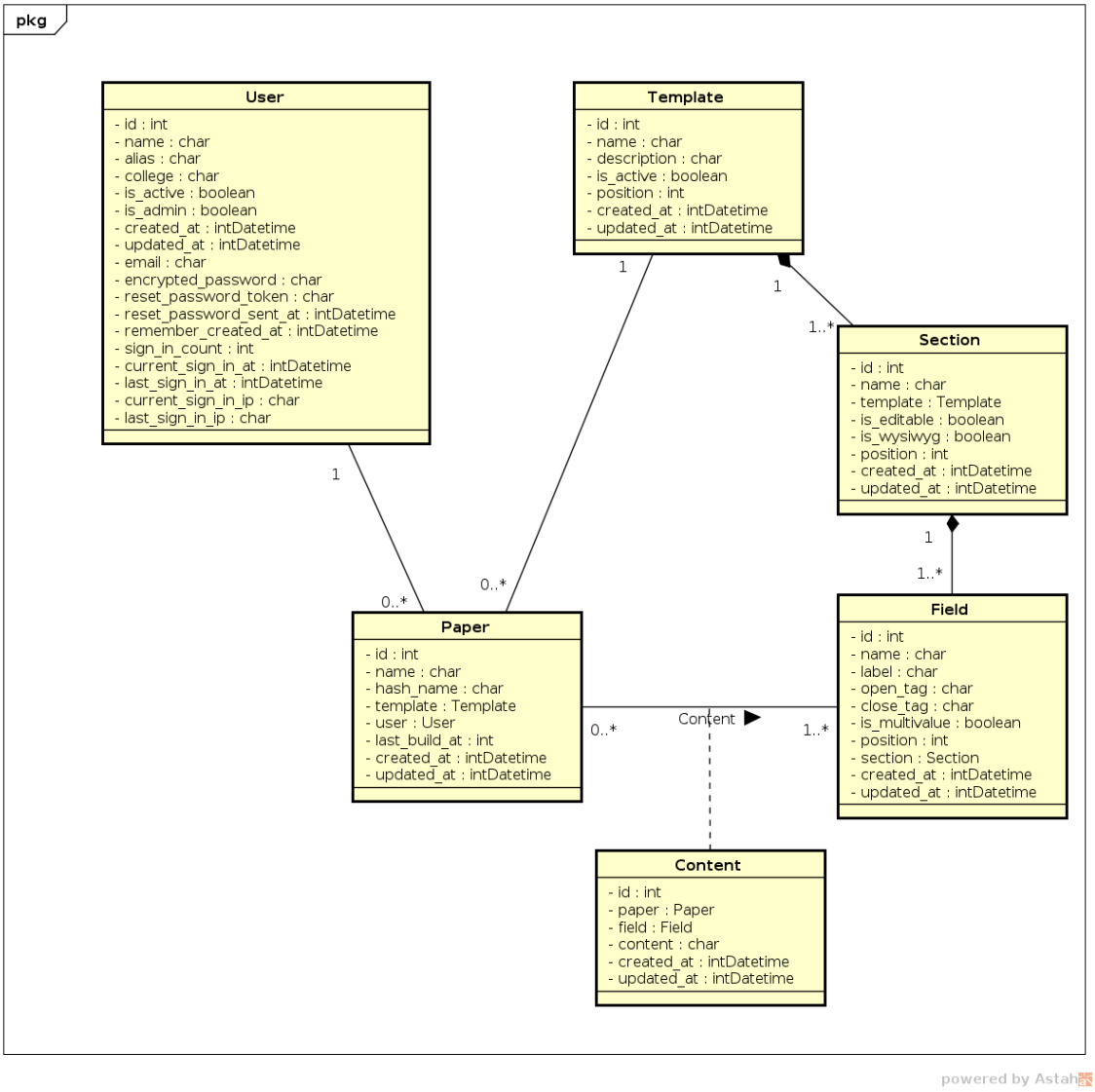



Fonte: os autores (2017)

*CRUD* é um acrônimo para as operações básicas de um banco de dados - *Create*, *Retrieve*, *Update* e *Delete*. Diversos Casos de Uso da Figura 63 representam esse conjunto de ações, identificados pela própria expressão.

# APÊNDICE D – DIAGRAMA DE CLASSES

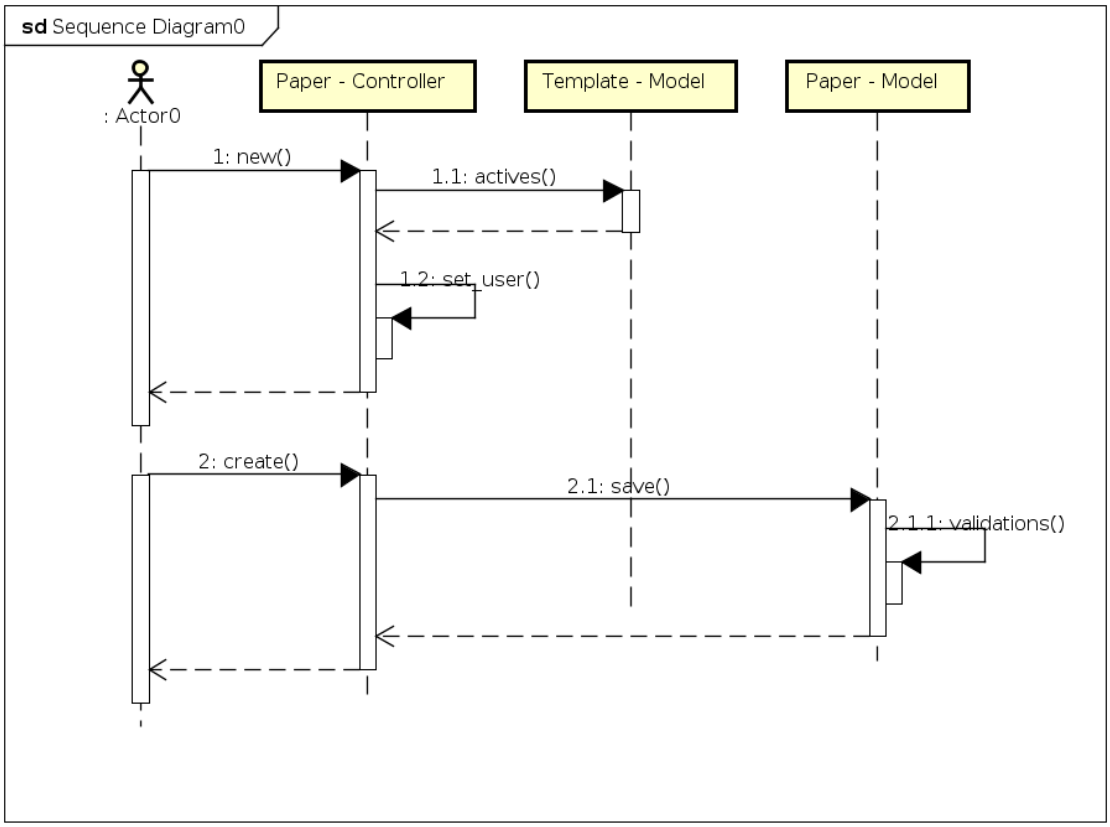
Figura 64 – Diagrama de Classes




Fonte: os autores (2017)

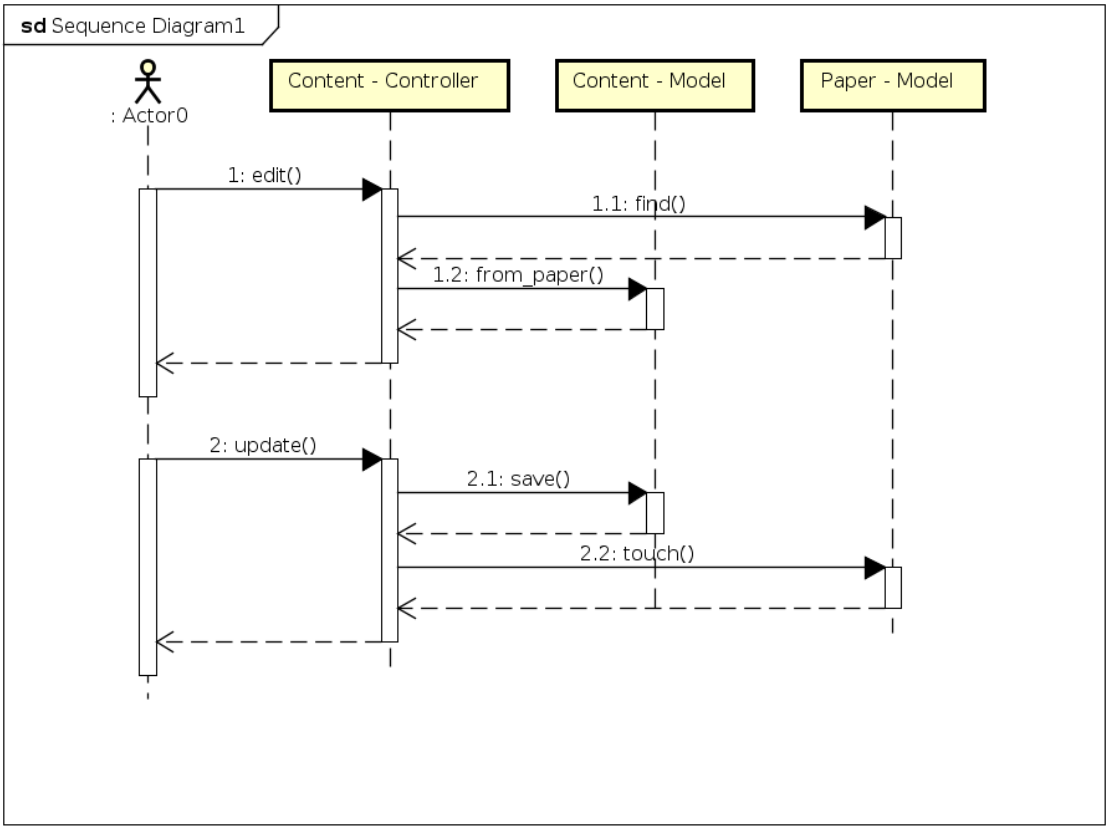
# APÊNDICE E – DIAGRAMAS DE SEQUÊNCIA

Figura 65 – Diagrama de Sequência: criação de trabalho



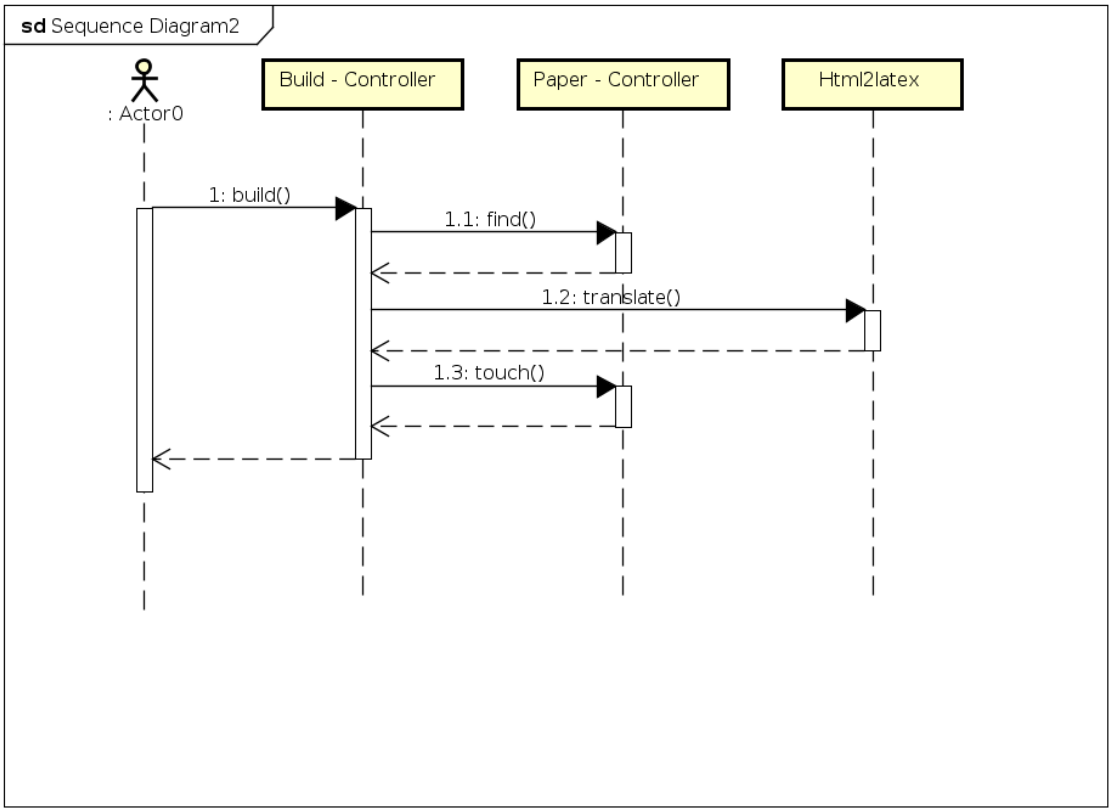

Fonte: os autores (2017)

Figura 66 – Diagrama de Sequência: edição de conteúdo




Fonte: os autores (2017)

Figura 67 – Diagrama de Sequência: conversão das tags




Fonte: os autores (2017)

# APÊNDICE F – ESPECIFICAÇÕES DE CASO DE USO

Tabela 5 – Especificação de caso de uso: criação de trabalho

| Nome do caso de uso | Criação de trabalho |
|---|---|
| Ator principal | Usuário |
| Resumo | Esse caso de uso descreve as etapas para o usuário realizar o cadastro de um novo trabalho |
| Pré-condições | Usuário precisa estar logado no sistema |
| Pós-condições | Usuário terá acesso ao trabalho criado e aos conteúdos do trabalho referentes ao *template* escolhido |
| Fluxo principal | |
| 1. O usuário acessa a tela de cadastrar trabalho<br>2. O sistema carrega os *templates* ativos<br>3. O sistema carrega o *select box*<br>4. O sistema carrega o usuário logado<br>5. O sistema apresenta a tela para o usuário<br>6. O usuário preenche o nome do trabalho<br>7. O usuário seleciona o *template*<br>8. O usuário clica no botão enviar<br>9. O sistema carrega o novo *template*<br>10. O sistema valida os dados do usuário (E1)<br>11. O sistema salva o novo trabalho<br>12. O sistema redireciona para a tela de trabalhos do usuário<br>13. O caso de uso é encerrado | |
| Fluxo de exceção 1 - Nome do trabalho já existe | |
| 1. O sistema carrega o formulário preenchido<br>2. O sistema exibe a mensagem de erro "ome do trabalho já existe" | |

Fonte: os autores (2017)

Tabela 6 – Especificação de caso de uso: edição de conteúdo

| Nome do caso de uso | Edição de conteúdo |
|---|---|
| Ator principal | Usuário |
| Resumo | Esse caso de uso descreve as etapas para o usuário editar o conteúdo de um trabalho |
| Pré-condições | Usuário precisa estar logado no sistema |
| Fluxo principal | |
| 1. O usuário acessa a tela de edição do conteúdo<br>2. O sistema carrega o trabalho<br>3. O sistema carrega o conteúdo da primeira seção do trabalho<br>4. O sistema carrega as informações nos campos apropriados<br>5. O sistema apresenta a tela para o usuário<br>6. O usuário preenche realiza as alterações nos campos disponíveis<br>7. O usuário clica no botão salvar (A1)<br>8. O sistema salva o novo conteúdo<br>9. O sistema atualiza a data de alteração do trabalho<br>10. O sistema reapresenta a tela para o usuário<br>11. O caso de uso é encerrado | |
| Fluxo alternativo 1 - O usuário seleciona outro conteúdo do trabalho | |
| 1. O sistema carrega o conteúdo da seção selecionada<br>2. O sistema carrega as informações nos campos apropriados<br>3. O sistema apresenta a tela para o usuário<br>4. O usuário preenche realiza as alterações nos campos disponíveis<br>5. O usuário clica no botão salvar | |

Fonte: os autores (2017) (2017)

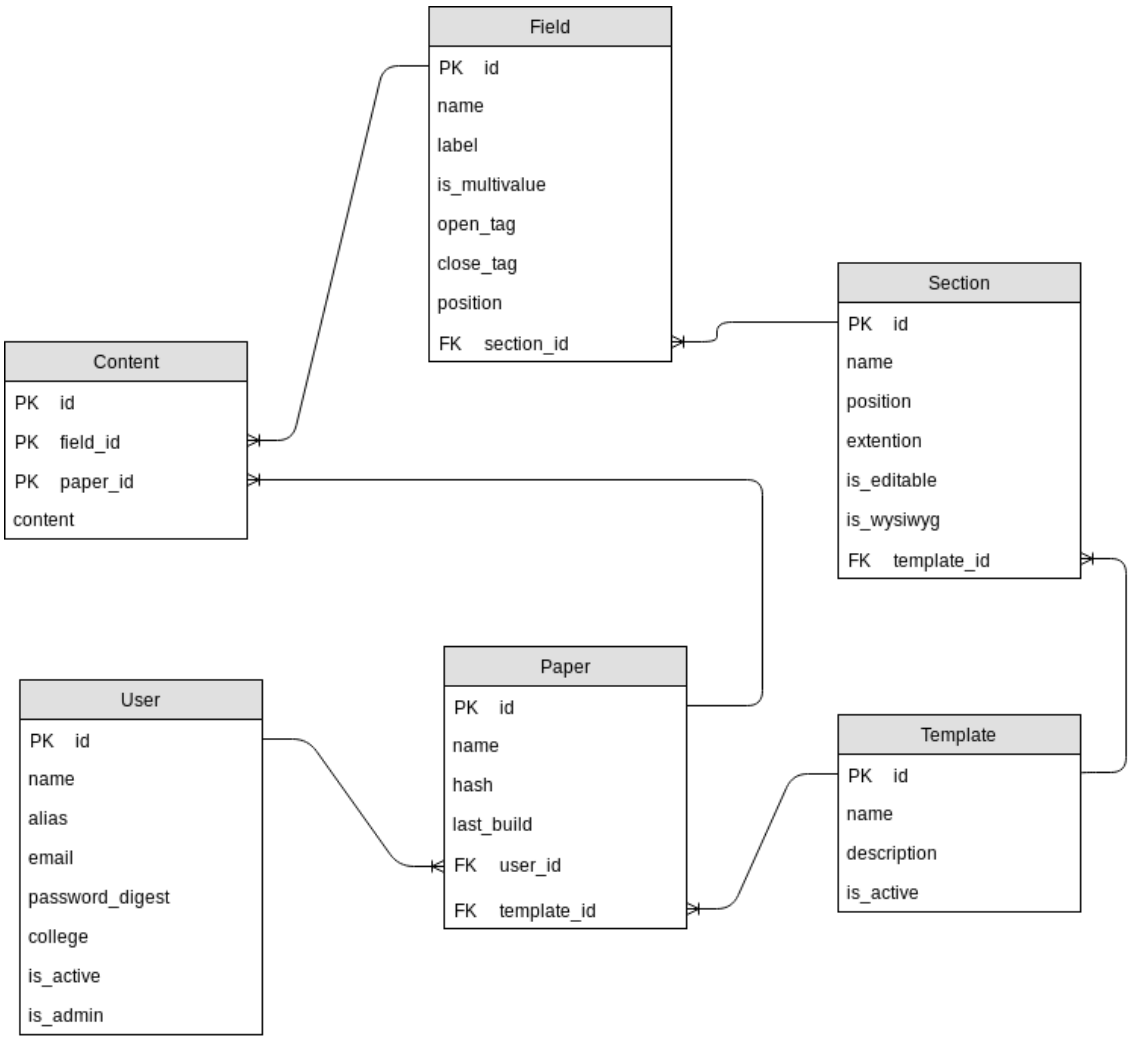
Tabela 7 – Especificação de caso de uso: conversão das tags

| Nome do caso de uso | conversão das tags |
|---|---|
| Ator principal | Usuário |
| Resumo | Esse caso de uso descreve as etapas para o usuário compílar um trabalho TeX e acessar seu PDF |
| Pré-condições | Usuário precisa estar logado no sistema |
| Pós-condições | Usuário terá acesso à versão PDF do trabalho compilado |
| Fluxo principal | |
| 1. O usuário acessa a tela de compilação do trabalho<br>2. O sistema carrega o trabalho<br>3. O sistema envia o conteúdo do trabalho para conversão HTML - LaTeX<br>4. O sistema atualiza a data de alteração do trabalho<br>5. O sistema salva o arquivo convertido em disco<br>6. O sistema chama o programa compilador passando o arquivo gerado como parâmetro<br>7. O sistema redireciona o navegador para o arquivo PDF gerado<br>8. O caso de uso é encerrado | |

Fonte: os autores (2017) (2017)

# APÊNDICE G – DER AJUSTADO

Figura 68 – Diagrama Entidade Relacionamento Ajustado




Fonte: os autores (2017)

ANEXOS